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
URL: http://www.kasahara.cs.waseda.ac.jp/

Waseda Univ. GCSC
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).

\[
\text{Power} \propto \text{Frequency} \times \text{Voltage}^2 \\
(\text{Voltage} \propto \text{Frequency})
\]

\[
\text{Power} \propto \text{Frequency}^3
\]

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

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

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”
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)

- 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 128 cores 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

Fjitsu M9000 SPARC Multicore Server

OSCAR Compiler gives us 211 times speedup with 128 cores

Commercial compiler gives us 0.9 times speedup with 128 cores (slow-downed against 1 core)
Trend of Peak Performances of Supercomputers

Aurora, 2018, 180PFLOPS, 13MW, Argonne National Lab., Intel & Cray

Sunway TaihuLight, 2016.06, 93PFLOPS, 15.4MW

Tianhe-2, 2013.06, 55PFLOPS, 17.8MW

Titan, 2012.11, 27PFLOPS, 8.2MW

Sequoia, 2012.06, 20PFLOPS, 7.9MW

京, 2011.6&11, 11PFLOPS, 11.3MW

2020-22 米中欧日 ExaFLOPS計画
Power Reduction of MPEG2 Decoding to 1/4 on 8 Core Homogeneous Multicore RP-2 by OSCAR Parallelizing Compiler

Avg. Power 5.73 [W] 73.5% Power Reduction Avg. Power 1.52 [W]
Compiler Co-designed Multicore RP2

On-chip system bus (SuperHyway)

LCPG: Local clock pulse generator
PCR: Power Control Register
CCN/BAR: Cache controller/Barrier Register
URAM: User RAM (Distributed Shared Memory)
Renesas-Hitachi-Waseda Low Power 8 core RP2
Developed in 2007 in METI/NEDO project

<table>
<thead>
<tr>
<th>Process Technology</th>
<th>90nm, 8-layer, triple-Vth, CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip Size</td>
<td>104.8mm² (10.61mm x 9.88mm)</td>
</tr>
<tr>
<td>CPU Core Size</td>
<td>6.6mm² (3.36mm x 1.96mm)</td>
</tr>
<tr>
<td>Supply Voltage</td>
<td>1.0V–1.4V (internal), 1.8/3.3V (I/O)</td>
</tr>
<tr>
<td>Power Domains</td>
<td>17 (8 CPUs, 8 URAMs, common)</td>
</tr>
</tbody>
</table>

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 in R&D and target practical applications

For smart life
- Solar Powered Smart phones
- Cameras
- Robots

Consumer electronic
- Internet TV/DVD
- Camcorders
- Capsule inner cameras

On-board vehicle technology
- (navigation systems, integrated controllers, infrastructure coordination)

Industry
- Intelligent home appliances
- Compiler, API
- Medical servers
- Heavy particle radiation planning, cerebral infarction

Green supercomputers
- Super real-time disaster simulation (tectonic shifts, tsunami), tornado, flood, fire spreading)
- Stock trading

OSCAR
- Many-core system technologies with ultra-low power consumption
- OSCAR many-core chip
- Many-core system technologies with ultra-low power consumption

OSCAR
- Green supercomputers
- Stock trading

OSCAR
- Protect lives
- Protect environment

Waseda University: R&D
- Many-core system technologies with ultra-low power consumption
- OSCAR many-core chip

National Institute of Radiological Sciences
- Operation/recharging by solar cells
- Non-fan, cool, quiet servers designed for server

For smart life
- On-board vehicle technology
- Consumer electronic
- Intelligent home appliances
- Green cloud servers

OSCAR
- Protect lives
- Protect environment

OSCAR
- Many-core system technologies with ultra-low power consumption
- OSCAR many-core chip

Industry
- Intelligent home appliances
- Compiler, API
- Medical servers
- Heavy particle radiation planning, cerebral infarction

Supercomputers and servers
- Super real-time disaster simulation (tectonic shifts, tsunami), tornado, flood, fire spreading)
- Stock trading
Cancer Treatment
Carbon Ion Radiotherapy
(Previous best was 2.5 times speedup on 16 processors with hand optimization)

8.9 times 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)
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

Compile Option:


(*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

---

**Diagram:**

- **Program**
  - **BPA**
  - **RB**
  - **SB**

  **Near fine grain parallelization**
  - Loop level parallelization
  - Near fine grain of loop body
  - Coarse grain parallelization

**Total System**

<table>
<thead>
<tr>
<th>Layer</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st. Layer</td>
<td>BPA, RB, SB</td>
</tr>
<tr>
<td>2nd. Layer</td>
<td>BPA, RB, SB</td>
</tr>
<tr>
<td>3rd. Layer</td>
<td>BPA, RB, SB</td>
</tr>
</tbody>
</table>
Earliest Executable Condition Analysis for Coarse Grain Tasks (Macro-tasks)

A Macro Flow Graph

A Macro Task Graph
PRIORITIZATION DETERMINATION IN DYNAMIC CP METHOD

Critical path length: \( 60 \times 0.80 + 100 \times 0.20 = 68 \)
# 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.

<table>
<thead>
<tr>
<th>Macrotask No.</th>
<th>Earliest Executable Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1 2</td>
</tr>
<tr>
<td>3</td>
<td>(1) 3</td>
</tr>
<tr>
<td>4</td>
<td>2 4 OR (1) 3</td>
</tr>
<tr>
<td>5</td>
<td>(4) 5 AND [ 2 4 OR (1) 3 ]</td>
</tr>
<tr>
<td>6</td>
<td>3 OR (2) 4</td>
</tr>
<tr>
<td>7</td>
<td>5 OR (4) 6</td>
</tr>
<tr>
<td>8</td>
<td>(2) 4 OR (1) 3</td>
</tr>
<tr>
<td>9</td>
<td>(8) 9</td>
</tr>
<tr>
<td>10</td>
<td>(8) 10</td>
</tr>
<tr>
<td>11</td>
<td>8 9 OR 8 10</td>
</tr>
<tr>
<td>12</td>
<td>11 12 AND [ 9 OR (8) 10 ]</td>
</tr>
<tr>
<td>13</td>
<td>11 13 OR 11 12</td>
</tr>
<tr>
<td>14</td>
<td>(8) 9 OR (8) 10</td>
</tr>
<tr>
<td>15</td>
<td>2 15</td>
</tr>
</tbody>
</table>

- **MT3 may start execution after MT1 branches to MT3.**
- **MT6 may start execution after MT3 finish execution or MT2 branches to MT4.**
- **MT2 may start execution after MT1 branches to MT2 and MT1 finish execution.**
Automatic processor assignment in 103.su2cor

- Using 14 processors

Coarse grain parallelization within DO400

\[ N_{PG}, N_{PE} = [PG, PE] \]
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
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 $\text{DGCI}{\text{R}}^{p}(1 \leq p \leq n)$
  - $n$: (multiple) num of PCs, $\text{DGCI}{\text{R}}$: Decomposed GCIR
- Generate CAR on which $\text{DGCI}{\text{R}}^{p}$ & $\text{DGCI}{\text{R}}^{p+1}$ are data-dep.
- Generate LR on which $\text{DGCI}{\text{R}}^{p}$ is data-dep.
Data Localization

MTG

MTG after Division

A schedule for two processors
An Example of Data Localization for Spec95 Swim

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

```
DO 200 J=1,N
DO 200 I=1,M
    UNEW(I+1,J) = UOLD(I+1,J)+
1   TDTS8*(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)-TDTS8*(H(I+1,J)-H(I,J))
    VNEW(I,J+1) = VOLD(I,J+1)-TDTS8*(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 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
```

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

Cache line conflicts occurs among arrays which share the same location on cache

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN</td>
<td>VN</td>
<td>PN</td>
<td>UO</td>
<td>VO</td>
</tr>
<tr>
<td>PO</td>
<td>CU</td>
<td>CV</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VN</td>
<td>PN</td>
<td>UO</td>
<td>VO</td>
</tr>
<tr>
<td>PO</td>
<td>U</td>
<td>V</td>
<td>P</td>
<td>UN</td>
</tr>
<tr>
<td>VN</td>
<td>PN</td>
<td>UO</td>
<td>UN</td>
<td></td>
</tr>
</tbody>
</table>

```
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 \(CU(N1,N2), CV(N1,N2),\)
* \(Z(N1,N2), H(N1,N2)\)

**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 \(CU(N1,N2), CV(N1,N2),\)
* \(Z(N1,N2), H(N1,N2)\)

Box: Access range of DLG0
Statement Level Near Fine Grain Task

<<LU Decomposition>>
1) \( u_{12} = \frac{a_{12}}{l_{11}} \)
2) \( u_{24} = \frac{a_{24}}{l_{22}} \)
3) \( u_{34} = \frac{a_{34}}{l_{33}} \)
4) \( l_{54} = -l_{52} \times u_{24} \)
5) \( u_{45} = \frac{a_{45}}{l_{44}} \)
6) \( l_{55} = u_{55} - l_{54} \times u_{45} \)

<<Forward Substitution>>
7) \( y_1 = \frac{b_1}{l_{11}} \)
8) \( y_2 = \frac{b_2}{l_{22}} \)
9) \( b_5 = b_5 - l_{52} \times y_2 \)
10) \( y_3 = \frac{b_3}{l_{33}} \)
11) \( y_4 = \frac{b_4}{l_{44}} \)
12) \( b_5 = b_5 - l_{54} \times y_4 \)
13) \( y_5 = \frac{b_5}{l_{55}} \)

<<Backward Substitution>>
14) \( x_4 = y_4 - u_{45} \times y_5 \)
15) \( x_3 = y_3 - u_{34} \times x_4 \)
16) \( x_2 = y_2 - u_{24} \times x_4 \)
17) \( x_1 = y_1 - u_{12} \times x_2 \)
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

Precedence relations

Flag set

Flag check

Unnecessary
Generated Parallel Machine Code for Near Fine Grain Parallel Processing

PE1

; Task A
; Task Body
FADD R23, R19, R21

; Task B
; Flag Check : Task C
L18:
LDR R28, [R14, 0]
CMP R28, R0
JNE L18

; Data Receive
LDR R29, [R14, 1]

; Task Body
FMLT R24, R23, R29

PE2

; Task C
; Task Body
FMLT R27, R28, R29
FSUB R29, R19, R27

; Data Transfer
STR [R14, 1], R29

; Flag Set
STR [R14, 0], R0
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 2 cores, 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.)
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

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

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

Generation of parallel machine codes using sequential compilers

Executable on various multicores

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

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

OSCAR: Optimally Scheduled Advanced Multiprocessor API: Application Program Interface
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.
Performance on Multicore Server for Latest Cancer Treatment Using Heavy Particle (Proton, Carbon Ion)

327 times speedup on 144 cores

Hitachi 144cores SMP Blade Server BS500:
Xeon E7-8890 V3(2.5GHz  18core/chip) x8 chip

- Original sequential execution time 2948 sec (50 minutes) using GCC was reduced to 9 sec with 144 cores (327.6 times speedup)
- Reduction of treatment cost and reservation waiting period is expected
110 Times Speedup against the Sequential Processing for GMS Earthquake Wave Propagation Simulation on Hitachi SR16000 (Power7 Based 128 Core Linux SMP)

Fortran: 15 thousand lines

First touch for distributed shared memory and cache optimization over loops are important for scalable speedup
Parallel Processing of JPEG XR Encoder on TILEPro64

Multimedia Applications:
- Sequential C Source Code
- Parallelized C Program with OSCAR API

OSCAR Compiler
Parallelized C Program with OSCAR API
API Analyzer + Sequential Compiler
Parallelized Executable Binary for TILEPro64

Cache Allocation Setting

Speedup (JPEG XR Encoder)
- 55x speedup on 64 cores

1. OSCAR Parallelization
2. Cache Allocation Setting

Local cache optimization:
Parallel Data Structure (tile) on heap allocating to local cache
Speedup with 2 cores for Engine Crankshaft Handwritten Program on RPX Multi-core Processor

1.6 times Speed up by 2 cores against 1 core

Macrotask graph with a lot of conditional branches

Branches are fused to macrotasks for static scheduling

Macrotask graph after task fusion

Grain is too fine (us) for dynamic scheduling.
Model Base Designed Engine Control on V850 Multicore with Denso

Though so far parallel processing of the engine control on multicore has been very difficult, Denso and Waseda succeeded 1.95 times speedup on 2core V850 multicore processor.

Hard real-time automobile engine control by multicore

C codes generated by MATLAB/Simulink embedded coder are automatically parallelized.
OSCAR Compile Flow for Simulink Applications

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)
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](http://www.mathworks.co.jp/jp/help/vision/examples)
Parallel Processing on Simulink Model

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

Call sequential C code from the S-Function block

Call parallelized C code from the S-Function block
OSCAR API Ver. 2.0 for Homogeneous/Heterogeneous Multicores and Manycores

List of Directives (22 directives)

- Parallel Execution API
  - parallel sections (*)
  - flush (*)
  - critical (*)
  - execution

- Memory 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

- 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

(* from OpenMP)

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)

Y. Yuyama, et al., "A 45nm 37.3GOPS/W Heterogeneous Multi-Core SoC", ISSCC2010
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

- Realtime processing mode with dead line constraints
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

<table>
<thead>
<tr>
<th>state</th>
<th>FULL</th>
<th>MID</th>
<th>LOW</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>0</td>
</tr>
<tr>
<td>voltage</td>
<td>1</td>
<td>0.87</td>
<td>0.71</td>
<td>0</td>
</tr>
<tr>
<td>dynamic energy</td>
<td>1</td>
<td>3/4</td>
<td>1/2</td>
<td>0</td>
</tr>
<tr>
<td>static power</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• State transition overhead

<table>
<thead>
<tr>
<th>state</th>
<th>FULL</th>
<th>MID</th>
<th>LOW</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>0</td>
<td>40k</td>
<td>40k</td>
<td>80k</td>
</tr>
<tr>
<td>MID</td>
<td>40k</td>
<td>0</td>
<td>40k</td>
<td>80k</td>
</tr>
<tr>
<td>LOW</td>
<td>40k</td>
<td>40k</td>
<td>0</td>
<td>80k</td>
</tr>
<tr>
<td>OFF</td>
<td>80k</td>
<td>80k</td>
<td>80k</td>
<td>0</td>
</tr>
</tbody>
</table>

delay time [u.t.]

<table>
<thead>
<tr>
<th>state</th>
<th>FULL</th>
<th>MID</th>
<th>LOW</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULL</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>MID</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>LOW</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>OFF</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

delay time [u.t.]

<table>
<thead>
<tr>
<th>state</th>
<th>FULL</th>
<th>MID</th>
<th>LOW</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>FULL</td>
<td>MID</td>
<td>LOW</td>
<td>OFF</td>
</tr>
<tr>
<td>FULL</td>
<td>0</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>MID</td>
<td>20</td>
<td>0</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>LOW</td>
<td>20</td>
<td>20</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>OFF</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>0</td>
</tr>
</tbody>
</table>

delay time [u.t.]

energy overhead [μJ]
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

Doall 6, Loop 10,11,12,13, Doall 17, Loop 18,19, 20, 21 are on CP
Low-Power Optimization with OSCAR API

Scheduled Result by OSCAR Compiler

VC0
MT1
MT2
Sleep
MT3
MT4
VC1

Generate Code Image by OSCAR Compiler

void main_VC0() {
  #pragma oscar fvcontrol 
  ((OSCAR_CPU(),0))
  Sleep
}

void main_VC1() {
  #pragma oscar fvcontrol 
  (1,(OSCAR_CPU(),100))
  MT1
  MT2
}

MT3
MT4

Generate Code Image by OSCAR Compiler

void main_VC0() {
  #pragma oscar fvcontrol 
  ((OSCAR_CPU(),0))
  Sleep
}

void main_VC1() {
  #pragma oscar fvcontrol 
  (1,(OSCAR_CPU(),100))
  MT1
  MT2
}

MT3
MT4

47
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]

1 cycle: 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/UCS43lNYEIkC8i_KIgFZYQBQ

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.97</td>
<td>0.63 (35.0%)</td>
<td>2/3 (-35.0%)</td>
</tr>
<tr>
<td>2</td>
<td>1.88</td>
<td>0.46 (-75.5%)</td>
<td>1/4 (-75.5%)</td>
</tr>
<tr>
<td>3</td>
<td>2.79</td>
<td>0.37 (-61.9%)</td>
<td>1/3 (-61.9%)</td>
</tr>
</tbody>
</table>

- 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 (Deadline 66.6 [ms/frame])

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

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

<table>
<thead>
<tr>
<th>number of PE</th>
<th>1PE</th>
<th>2PE</th>
<th>3PE</th>
</tr>
</thead>
<tbody>
<tr>
<td>average power consumption [W]</td>
<td>29.29</td>
<td>36.59</td>
<td>41.58</td>
</tr>
<tr>
<td>without power control</td>
<td>24.17</td>
<td>12.21</td>
<td>9.60</td>
</tr>
<tr>
<td>with power control</td>
<td>24.17</td>
<td>12.21</td>
<td>9.60</td>
</tr>
</tbody>
</table>
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

![Graph showing speed-ups on TILEPro64 Manycore](image)
OSCAR Vector Multicore and Compiler for Embedded to Servers with OSCAR Technology

Target:
- Solar Powered with compiler power reduction.
- Fully automatic parallelization and vectorization including local memory management and data transfer.
Fujitsu VPP500/NWT: PE Unit
Faster or Equal Processing Performance up to 4 cores with hardware coherent mechanism on RP2.

Software Coherence gives us correct execution without hardware coherence mechanism on 8 cores.
Automatic Local Memory Management
Data Localization: Loop Aligned Decomposition

• Decomposed loop into LR(s) and CAR(s)
  – 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

```
DO I=1,101
  A(I)=2*I
  ENDDO

DO I=1,100
  B(I)=B(I-1)
  +A(I)+A(I+1)
  ENDDO

DO I=2,100
  C(I)=B(I)*B(I-1)
  ENDDO
```
Adjustable Blocks

• Handling a suitable block size for each application
  – different from a fixed block size in cache
  – each block can be divided into smaller blocks with integer divisible sizes to handle small arrays and 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] -> TA[offset + i’ * L + j’ * M + k’]
  - Template Arrays provide readability
    - A[i][j][k] -> TA[Block#][i’][j’][k’]
8 Core RP2 Chip Block Diagram

On-chip system bus (SuperHyway)

LCPG: Local clock pulse generator
PCR: Power Control Register
CCN/BAR: Cache controller/Barrier Register
URAM: User RAM (Distributed Shared Memory)
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**
Automatic Software Coherent Control for Manycores

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

![Graph showing the performance comparison between SMP (Hardware Coherence) and NCC (Software Coherence). The x-axis represents the number of processor cores (1 to 8), and the y-axis represents Speedup. The graph includes applications from SPEC2000, SPEC2006, NPB, MediaBench, and MPEG2 Encoder.]
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
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 fun, 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 earth quakes
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 144 cores 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 Tilera 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.