OSCAR API for Low-Power Multicores and Manycores, and API Standard Translator

Keiji Kimura, Waseda University
Current Multicores and Their Architecture

- Multicores are Everywhere Now!
  - Server, Desktop, and Embedded
- Multicores for Server and Desktop Usage
  - Ordinary Shared Memory Architecture
- Multicores for Embedded Usage
  - Various Kinds of Memory Architecture
    - Local Memory (Scratch Pad Memory), Distributed Shared Memory, On-chip/Off-chip Shared Memory, …
- Power Control Mechanisms
  - DVFS, Clock Gating, Power Gating, …
Current Application Development Environment for Multicores

- Parallel API
  - pthread (SMP)
    - Old thread library
  - OpenMP (Shared Memory), MPI (Distributed Memory)
    - for Scientific Applications
  - Co-array Fortran (PGAS), UPC (PGAS)
    - Language extension
  - MCAPI (Distributed Memory)
    - for Embedded Applications
    - Message Passing API

- NO Good API for Low-Power and Real-time Multicores!

- Parallelizing Compilers for Scientific Applications (Fortran Applications)
  - Several aggressive compilers from academia
    - Polaris, SUIF, CETUS, Pluto, (OSCAR), …
  - Source-to-source Compiler

- Parallelizing Compilers for Low-Power and Real-time Multicores?
Our Effort with OSCAR Compiler

- **OSCAR Compiler**
  - Multigrain Parallel Processing
  - Data Locality Optimization
  - Data Transfer Optimization
  - Low-Power Optimization
  - **Good for Embedded Applications in addition to Scientific Applications**

- **OSCAR API**
  - Parallel API for Low-power and Real-time Multicores
    - Developed by CATS, DENSO, e-SOL, Fujitsu, Fujitsu Laboratory, GAIO Technology, Hitachi, MITSUBISHI Electric, NEC, Olympus, Panasonic, Renesas Electronics, Renesas Solutions, Toshiba, Toho University, Nagoya University and Waseda University in METI/NEDO Project
  - Supporting various kinds of memory architectures
    - Local Memory (Scratch Pad Memory), Distributed Shared Memory, On-chip/Off-chip Centralized Shared Memory
  - Supporting power control mechanism
    - DVFS, Clock Gating, Power Gating
  - Supporting accelerators and many-cores
    - from Version 2.0
  - **Using as an interface between OSCAR Compiler and various Multicores**
  - **Now Open! (Download from http://www.kasahara.cs.waseda.ac.jp/)**
OSCAR Automatic Parallelizing Compiler

Multigrain Parallel Processing
- Hierarchical and Global Parallelization
  - Coarse grain task parallel
  - Loop iteration parallel
  - Statement level parallel

Data Locality Optimization
- Task (or loop) decomposition considering cache size or local memory size
- Task scheduling considering data affinity

Low power optimization
- Power scheduling with DVFS and power gating by software

Task level or statement level parallelization
Application Development Environment with OSCAR Compiler, OSCAR API and API Standard Translator

OSCAR Compiler
- Multigrain parallel processing
- Data locality optimization
- Data Transfer optimization
- Low power optimization

Sequential Program

Written in Fortran or Parallelizable C

Parallelized Fortran or C code with OSCAR API 2.0

OpenMP Compiler

API Translator

Native Compiler

Backend Compiler

Exec. Object

RP2
Other Multicores

Parallelized Fortran or C code with OSCAR API 2.0
Overview of OSCAR API v2.0

- Targeting mainly real-time consumer electronics devices
  - Embedded computing
  - Various kinds of memory architecture
    - SMP, local memory, distributed shared memory, non-coherent cache ...
  - Power control mechanisms
  - Accelerators

- Based on the subset of OpenMP
  - Very popular parallel processing API
  - Shared memory programming model
  - Supporting both of C and Fortran

- Eight Categories
  - Parallel Execution
  - Memory Mapping
  - Data Transfer
  - Power Control
  - Timer
  - Synchronization
  - Accelerator
  - Cache Control
OSCAR Multicore Architecture

A Target Multicore doesn’t have to equip with all modules.
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

- **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 OpenMP)

from V2.0
int flag1_1;  
int myver;  
#pragma omp threadprivate(myver)  
int main() {  
#pragma omp parallel sections  
{  
#pragma omp section  
{ main_VC0(); }  
#pragma omp section  
{ main_VC1(); }  
...

#pragma omp section  
#pragma omp oscar \  
distributedshared(flag1_1)  
{ main_VC4(); }  
...
  }  

return 0;  
}  

void main_VC0() {  

MT1_1  
for Core0  

#pragma omp flush  
flag1_1 = myver;  
#pragma omp flush

MT1_2  
for Core0  

...

void main_VC1() {  

MT1_1  
for Core1  

void main_VC4() {  

MT1_1  
for Core1  

/* waiting for signal from VC0 */  
do {  
#pragma omp flush  

MT1_2  
for Core1  

MT1_3  
for Core4  

while (flag1_1  
!= myver);  

...  

}  

Generated code image by OSCAR Compiler
Image of Low-Power Optimization with OSCAR API

Scheduled Result by OSCAR Compiler

void main_VC0() {
    MT1
    MT2
    Sleep
    MT3
    MT4
}

Generate Code Image by OSCAR Compiler

void main_VC1() {
    MT2
    #pragma oscar fvcontrol \ ((OSCAR_CPU(),0))
    Sleep
    MT2
    #pragma oscar fvcontrol \ (1,(OSCAR_CPU(),100))
    MT3
    MT4
}
Accelerator Library

Compile flow for Heterogeneous Multicores

Source program

Compiler for ACCa

Source with Hint directives

Compiler for ACCb

Source with Hint directives

... Compiler for ACCz

Source with Hint directives

Chip Configuration Information

OSCAR Compiler

for CPU and each ACCs

for CPUs (C or Fortran)

for ACCa_0 functions

for ACCb_0 functions

Compiler for ACCa

Compiler for ACCb

Compiler for ACCz

for ACCz_0 functions

Compiler for ACCz

ACCa_0 object

ACCa_0 control code

ACCb_0 object

ACCb_0 control code

ACCz_0 object

ACCz_0 control code

API Translator

CPU code

Backend Compiler

Linker

Executable object

Accelerator Library

Inserting hint directives for OSCAR Compiler

Hint directives

• #pragma oscar_hint
• accelerator_task

Heterogeneous directives

• #pragma oscar

• accelerator_task_entry

12/07/11
“task_func” function is annotated with “oscar_hint” directive.

- loop1, loop2, loop3 can be compiled by the compiler for ACCa.
- func1 is a library function for ACCa.

```
main() {
    int a, b[10];
    #pragma oscar_hint accelerator_task
    (ACCa) cycle(1000) in(a, b[0:9])
    out(b[0:9])
    task_func();
}
task_func() {
    for (...) {...}// loop1
    for (...) {...}// loop2
    #pragma oscar_comment “XXXXX”
    func1 (...) // func1
    for (...) {...}// loop3
}
```

```
Sample.omp.c
main() {
    #pragma omp parallel sections
    {
        #pragma omp section
        {
            /* VC0, VPC 0 */
            oscartask_CTRL0_task_func();
            ...
        }
    }
}
```

```
Sample.VC1.c
#pragma oscar accelerator_task_entry
controller(0)
oscartask_CTRL0_task_func
oscartask_CTRL0_task_func() {
    for (...) {...}// loop1
    for (...) {...}// loop2
    #pragma oscar_comment “XXXXX”
    oscarlib_CTRL0_ACCEL1_func1 (...) // func1
    for (...) {...}// loop3
```
Cache Control API

- **cache_writeback**
  - Write back an indicated data on a cache to an off-chip memory

- **cache_selfinvalidate**
  - Invalidate an indicated data from a cache

- **complete_memop**
  - Wait for preceding memory operation on a processor core

- **noncacheable**
  - Locate an indicated data onto non-cacheable area

- **aligncache**
  - Align an indicated data onto a cache line of the last-level-cache
API Standard Translator

1. C/Fortran Program with OSCAR API
2. OSCAR API Standard Translator
3. Backend Compiler (gcc, etc)
4. Parallelized Execution Binary
5. Stub Library
6. Configuration file

- Stub Library: for platform runtime library, such as pthread library
- Configuration file: for platform configuration, such as memory map, available features, etc.
Translation Example

```c
int myver;
#pragma omp threadprivate(myver)
int main() {
#pragma omp parallel sections
{
#pragma omp section
{ main_VC0(); }
#pragma omp section
{ main_VC1(); }
...  
#pragma omp section
{ main_VC4(); }
...  
}
return 0;
}

int __attribute__((section("OSCAR_LDM")))myver;

int main() {
    int thr1, ..., thr4;
    oscar_thread_create(&thr1, thread_function_001, 0);
    ...
    oscar_thread_create(&thr3, thread_function_004, 3);
    thread_function_000();
    oscar_thread_join(thr1);
    ...
    oscar_thread_join(thr4);
    ...
    return 0;
}

void thread_function_000(void)
{
    main_VC0();
}

... MPSoC2012/Keiji Kimura 12/07/11 ...
```
Evaluations

- Scalability Evaluation
  - IBM p5 550Q (Power5+ x 4chips: 8 cores)
  - Renesas, Hitachi, Waseda RP2 (SH4A 8 cores: 4 cores are used)
    - Parallelizable C Applications
  - IBM p6 595 (Power6 x 16chips: 32 cores)
    - SPEC 95, 2000 fp Applications (Fortran)

- Power Optimization Evaluation
  - RP2 with 8 cores
  - AAC Encoder available on the market from Renesas Technology
  - MPEG2 Decoder from MediaBench (modified into Parallelizable C)
  - Real-time Execution Mode
Scalability Evaluation on IBM p5 550Q

5.8 times speedup on average
Scalability Evaluation on IBM p6 595

7.3 time speedup on average
3.3 times acceleration over XL Fortran
Consumer Electronics Multicore: RP2

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Technology</td>
<td>90nm, 8-layer, triple-Vth, CMOS</td>
</tr>
<tr>
<td>Chip Size</td>
<td>104.8mm$^2$</td>
</tr>
<tr>
<td></td>
<td>(10.61mm x 9.88mm)</td>
</tr>
<tr>
<td>CPU Core Size</td>
<td>6.6mm$^2$</td>
</tr>
<tr>
<td></td>
<td>(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>
Scalability Evaluation on RP2

![Graph showing speedup ratios for different applications and core counts.]

- Speedup ratio for 179.art: 1.61, 2.54, and 2.9 times with 1, 2, and 4 cores respectively.
- Speedup ratio for 183.equake: 1.40, 2.54, and 2.9 times with 1, 2, and 4 cores respectively.
- Speedup ratio for mpeg2encode: 1.73, 3.27, and 2.9 times with 1, 2, and 4 cores respectively.
- Speedup ratio for AACencode: 1.81, 3.35, and 2.9 times with 1, 2, and 4 cores respectively.

*2.9 times speedup on average*
Low-Power Optimization and OSCAR API on MPEG2 decoder

Without Power Control (Voltage: 1.4V)

With Power Control (Frequency, Resume Standby: Power shutdown & Voltage lowering 1.4V-1.0V)

76.0% Power Reduction

Avg. Power 5.41 [W]  1.30 [W]
Low-power consumer electronics heterogeneous multicore “RP-X”

Cluster #0
- SH-4A
- SHwy#0 (Address=40, Data=128)
  - DBSC #0
  - DMAC #0
  - VPU5
  - SHPБ
- SHwy#2 (Address=32, Data=64)
  - PCI exp
  - SATA
  - SPU2
  - LBSC
  - HPB

Cluster #1
- SH-4A
- SHwy#1 (Address=40, Data=128)
  - FE #0-3
  - DMAC #1
  - DBSC #1
  - MX2 #0-1

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

developed by Renesas, Hitachi, Titech, and Waseda
Scalability of AAC

<table>
<thead>
<tr>
<th></th>
<th>Homogeneous</th>
<th>Heterogeneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>1SH</td>
<td>1</td>
<td>16.08</td>
</tr>
<tr>
<td>2SH</td>
<td>1.98</td>
<td>8.39</td>
</tr>
<tr>
<td>4SH</td>
<td>3.68</td>
<td>4.44</td>
</tr>
<tr>
<td>8SH</td>
<td>6.32</td>
<td>3.61</td>
</tr>
<tr>
<td>1SH+1FE</td>
<td>3.61</td>
<td></td>
</tr>
<tr>
<td>2SH+1FE</td>
<td>4.44</td>
<td></td>
</tr>
<tr>
<td>4SH+2FE</td>
<td>8.39</td>
<td></td>
</tr>
<tr>
<td>8SH+4FE</td>
<td>16.08</td>
<td></td>
</tr>
</tbody>
</table>

Speedups against a single SH processor
Scalability of Optical Flow

Speedups against a single SH processor

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Speedup (fps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1SH</td>
<td>3.5</td>
</tr>
<tr>
<td>2SH</td>
<td>2.29</td>
</tr>
<tr>
<td>3SH</td>
<td>3.09</td>
</tr>
<tr>
<td>4SH</td>
<td>5.4</td>
</tr>
<tr>
<td>5SH</td>
<td>18.85</td>
</tr>
<tr>
<td>6SH+1FE</td>
<td>26.71</td>
</tr>
<tr>
<td>7SH+2FE</td>
<td>32.65</td>
</tr>
</tbody>
</table>

Homogeneous vs. Heterogeneous

MPSoc2012/Keiji Kimura 12/07/11
Low power optimization on Optical Flow

Without Power Reduction

With Power Reduction by OSCAR Compiler

70% of power reduction

Average: 1.76[W] → 0.54[W]

1 cycle: 33[ms] → 30[fps]
Conclusions

- **OSCAR API v2.0**
  - works with OSCAR compiler
  - supporting various kinds of memory architecture
    - Local memory, Distributed shared memory, on-chip shared memory, coherent cache, and non-coherent cache
  - supporting frequency and voltage control, per-core clock-off and per-core shutdown
  - supporting accelerators

- **OSCAR API standard translator**
  - Portable translator for various platforms

- The combination of OSCAR compiler, OSCAR API and the standard translator give us scalable performance improvement and low-power over various multicores.

- **OSCAR API specification is available from our web site.**