Multi-platform Automatic Parallelization and Power Reduction by OSCAR Compiler

Hironori Kasahara
Professor, Dept. of Computer Science & Engineering
Director, Advanced Multicore Processor Research Institute
Waseda University, Tokyo, Japan

IEEE Computer Society Board of Governors
IEEE Computer Society Multicore STC Chair
URL: http://www.kasahara.cs.waseda.ac.jp/
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.
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

Parallelized API F or C program
Proc0
Code with directives Thread 0
Proc1
Code with directives Thread 1
Accelerator 1
Code
Accelerator 2
Code

Low Power Homogeneous Multicore Code Generation
API Analyzer
Existing sequential compiler

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

Server Code Generation
OpenMP Compiler

Homegeneous Multicores from Vendor A
(SMP servers)

Heterogeneous Multicores from Vendor B

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

OSCAR: Optimally Scheduled Advanced Multiprocessor API: Application Program Interface

Execute on various multicores

Shred memory servers

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

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

Manual parallelization / power reduction

Low Power Homogeneous Multicore Code Generation
API Analyzer
Existing sequential compiler

Generation of parallel machine codes using sequential compilers

Proc0
Code with directives Thread 0
Proc1
Code with directives Thread 1
Accelerator 1
Code
Accelerator 2
Code
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.
Parallelizing Handwritten Engine Control Programs on Multi-core processors

- Current automotive crankshaft program
  - Developed by TOYOTA Motor Corp
  - About 300,000 Lines
  - Difficulty of parallel processing
    - Too fine granularity
    - Many conditional branches and small basic blocks, but **no parallelizable loops**
      - Minimizing run-time overhead and improvement of parallelism are necessary
        - Current product compilers can not parallelize
        - Current accelerators are not applicable

- Automatic parallelization of a crankshaft program using multi-grain parallelization in OSCAR Compiler
  - Performance improvement and efficient multi-threaded programming development
Analysis of Coarse Grain Parallelism by OSCAR Compiler

- Decomposes a program into coarse grain tasks, or macro tasks (MTs)
  1. BB (Basic Block)
  2. RB (Repetition Block, or loop)
  3. SB (Subroutine Block, or function)

- Generate MFG (Macro Flow Graph)
  - Control flow and data dependencies

- Generates MTG (Macro Task Graph)
  - Coarse grain parallelism

Data Dependency

Control Flow

Conditional Branch

Earliest Executable Condition

Macro-Flow Graph

Macro-Task Graph
Coarse Grain Task Parallelization of Hand-written Engine Control Program

- **Loop parallelization**
  - No parallelizable loops in engine control codes

- **Fine grain parallelization**
  - Each BBs are very low cost - less than 100 clock cycles
  - Branches prevent compilers

- **Coarse grain parallelization**
  - Utilize parallelism between SBs and BBs
Static Task Scheduling

- Dynamic task scheduling
  - Prevent from traceability
  - Add run-time overhead

- Static task scheduling
  - Guarantee Real-time constraints
  - Ensure traceability
  - Minimize run-time overhead

- Cannot assign BBs having branches statically
  - Static task scheduling can be applied if the MTG has only data dependency
  - The compiler cannot see if the branch is taken or not at compile time.

- Fuse tasks by hiding conditional branches in MFG to avoid dynamic task scheduling
  - Macro Task Fusion
Analysis of A Crankshaft Program Using Macro Task Fusion

There is not enough parallelism

MTG of crankshaft program before macro task fusion

Can not schedule MTs at compile time

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

MTG of crankshaft program after macro task fusion

There is not enough parallelism
MTG of Crankshaft Program Using Inline Expansion and Duplicating If-statements

MTG of crankshaft program before restructuring

- Succeed to reduce CP
  - 99% -> 60%

MTG of crankshaft program after restructuring

Successfully increased coarse grain parallelism

CP accounts for over 99% of whole execution time.

Critical Path (CP)

CP accounts for about 60% of whole execution time.

Critical Path (CP)
Evaluation Environment:
Embedded Multi-core Processor RPX

- SH-4A 648MHz * 8
  - As a first step, we use just two SH-4A cores because target dual-core processors are currently under design for next-generation automobiles.
Evaluation of Crankshaft Program with Multi-core Processors

• Attain 1.54 times speedup on RPX
  – There are no loops, but only many conditional branches and small basic blocks and difficult to parallelize this program

• This result shows possibility of multi-core processor for engine control programs
Performance of OSCAR Compiler on Intel Core i7 Notebook PC

• OSCAR Compiler accelerate Intel Compiler about 2.0 times on average

CPU: Intel Core i7 3720QM (Quad-core)
MEM: 32GB DDR3-SODIMM PC3-12800
OS: Ubuntu 12.04 LTS
Parallel Processing of JPEG XR Encoder on TILEPro64

Multimedia Applications:
- Sequential C Source Code
- AAC Encoder
- JPEG XR Encoder
- Optical Flow Calc.

OSCAR Compiler

Parallelized C Program with OSCAR API

API Analyzer + Sequential Compiler

Parallelized Executable Binary for TILEPro64

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

Local cache optimization:
Parallel Data Structure (tile) on heap allocating to local cache
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.
92 Times Speedup against the Sequential Processing for GMS Earthquake Wave Propagation Simulation on Hitachi SR16000 (Power7 Based 128 Core Linux SMP)
Profile-Based Automatic Parallelization and Sequential Program Tuning for Android 2D Rendering on Nexus7

**Android 2D Rendering “Skia”**
Standard library which performs 2D rendering on Android
Transform figure to some paths

**Profile-Based Automatic Parallelization**
Original Source Code

### Tuning Method for “Skia” [DrawRect]

SEOUL: motherboard 10400x10400 Int width. Int height.

 ```c
 void SkRGB16_Blitter::blitRectTextbox(int x, int y, int width, int height) {
  SkASSERT(x + width <= fDevice.width() && y + height <= fDevice.height());
  uint16_t* device = fDevice.getAddr16(x, y);
  unsigned deviceRB = fDevice.rowBytes();
  SkPMColor src = fSrcColor32;
  while (--height >= 0) {
    if (deviceRB == 0x1234) {
      device = (uint16_t*)((char*)device + deviceRB);
    }
    blend32_16_row(src, device, width);
  }
}
```

### Original Source Code

#### ARM Compiler (GCC)

#### Parallelizing Hotspot Information

#### OSCAR Parallelization Compiler

#### Parallelized Source Files with OSCAR API

#### ARM Compiler (GCC)

#### Parallelized Binary File

**Profile Result**

### Android 2D Rendering “Skia” Profiling

- **Update 1:** Android 2D Rendering “Skia”
- **Update 2:** Skia
- **Update 3:** Multicore

**Multicore “Skia” Profiling**

- **Zoom:** 50.77x
- **Zoom:** 50.77x
- **Zoom:** 50.77x
- **Zoom:** 50.77x

**Android 2D Rendering “Skia” Execution**

- **Finish 8sec**
- **Finish 8sec**
- **Finish 8sec**
- **Finish 8sec**

**Android 2D Rendering “Skia” Evaluations**

- **CPU Load Graph [DrawRect]***
- **CPU Load Graph [DrawRect]***
- **CPU Load Graph [DrawRect]***
- **CPU Load Graph [DrawRect]***

**Evaluations**

- **Sequential Skia – 1PE**
- **Parallelized Skia – 3PE**

### Evaluations

- **Sequential “Skia” – 1PE**
- **Parallelized “Skia” – 3PE**

**Performance**

- **DrawRect**
- **DrawCircle2**

**GPU Load Graph [DrawRect]**

- **22.82**
- **38.58**
- **33.86**
- **27.16**
- **1.91**

**CPU Load Graph [DrawRect]**

- **50.77**
- **50.77**
- **50.77**
- **50.77**

**Benchmarks**

- **DrawRect**
- **DrawCircle2**

**Conclusion**

- **OSCAR Compiler**
- **Skia**
- **Multicore**

**Green Computing Systems Research and Development Center Waseda University**
Parallelization of 2D Rendering Engine SKIA on 3 cores of Google NEXUS7

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

On Nexus7, 3 core parallelization gave us

- **for DrawRect**: \( \times 1.91 \) speedup
- **for DrawImage**: \( \times 1.95 \) speedup
void main_VC0() {
    MT1
    MT2
    #pragma oscar fvcontrol ¥ ((OSCAR_CPU(),0))
    Sleep
    MT3
    MT4
}

void main_VC1() {
    MT2
    #pragma oscar fvcontrol ¥ ((OSCAR_CPU(),100))
    Sleep
    MT3
    MT4
}
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)

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

Avg. Power
5.73 [W]  73.5% Power Reduction  1.52 [W]
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 3.7-3.0GHz Heterogeneous Multi-Core SoC", ISSCC2010

Speedups against a single SH processor:
- 1SH: 1 fps
- 2SH: 2.29 fps
- 4SH: 3.09 fps
- 8SH: 5.4 fps
- 2SH+1FE: 18.85 fps
- 4SH+2FE: 26.71 fps
- 8SH+4FE: 32.65 fps

111 fps
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

With Power Reduction by OSCAR Compiler

70% of power reduction

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>Number of Processor Cores</th>
<th>Power Consumption [W]</th>
<th>Without Power Reduction</th>
<th>With Power Reduction</th>
<th>Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.97</td>
<td>0.63</td>
<td>2/3 (-35.0%)</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>1.88</td>
<td>0.46</td>
<td>1/4 (-75.5%)</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2.79</td>
<td>0.37</td>
<td>1/7 (-86.7%)</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.
Automatic Power Reduction on 4 core Intel Haswell

- Haswell Processor
  - OS Ubuntu 13.10
  - Intel CPU Core i7 4770K
    - 4 cores
    - L1 Cache: Load 64Bytes/cycle, Store 32Bytes/cycle
    - L2 Cache 64Bytes/cycle
    - L3 Cache 8 MB
    - Frequency 3.5GHz~0.8MHz
  - Memory 16GB (8GB×2)
Power Reduction on Intel Haswell for Real-time Optical Flow

Power was reduced to 1/4 by the compiler power optimization on the same 3 cores.
The power with 3 core was reduced to 1/3 against 1 core.
Power Waves for 1 Core to 3 Cores without the Compiler Power Control on Intel Haswell for Real-time Optical Flow

29.29W

36.59W

41.58W

1 Core

2 Cores

3 Cores
Power Waves for 1 Core to 3 Cores with the Compiler Power Control on Intel Haswell for Real-time Optical Flow
Power for 1 & 3 Cores without Control vs. for 3 Cores with Control on Haswell

Without Power Control

With Power Control

1 Core

3 Cores

3 Cores
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