The Global Control-Flow Graph
Optimizing an Event-Driven Real-Time System
Across Kernel Boundaries

Christian Dietrich, Martin Hoffmann, Daniel Lohmann
{dietrich,hoffmann,lohmann}@cs.fau.de

Friedrich-Alexander University
Erlangen-Nuremberg

Compiler Optimization on Function Level

```c
void compute(int a[], int len, int val) {
    for (int i = 0; i < len; i++) {
        a[i] = a[i] + val + 1000;
    }
}
```
void compute(int a[], int len, int val) {
    for (int i = 0; i < len; i++) {
        a[i] = a[i] + val + 1000;
    }
}

Calculated in each Iteration
Compiler Optimization on Function Level

void compute(int a[], int len, int val) {
    for (int i = 0; i < len; i++) {
        a[i] = a[i] + val + 1000;
    }
}

Calculated in each Iteration

Loop-Invariant Code Motion

void compute(int a[], int len, int val) {
    int temp = val + 1000;
    for (int i = 0; i < len; i++) {
        a[i] = a[i] + temp;
    }
}
int lastVal, data[2];
void compute(int a[], int len, int val) {
    int temp = val + 1000;
    ...
}
void Task1() {
    compute(data, 2, lastVal);
}
Compiler Optimization on Program Level

```c
int lastVal, data[2];
void compute(int a[], int len, int val) {
    int temp = val + 1000;
    ...
}
void Task1() {
    compute(data, 2, lastVal);
}
```
Compiler Optimization on Program Level

```c
int lastVal, data[2];
void compute(int a[], int len, int val) {
    int temp = val + 1000;
    ...
}
void Task1() {
    compute(data, 2, lastVal);
}
```

Inlining and Loop Unrolling

```c
int lastVal, data[2];
void Task1() {
    int temp = lastVal + 1000;
    data[0] = data[0] + temp;
}
```
void Task1()
{
    int temp = lastVal + 1000;
    data[0] = data[0] + temp;
}

void Task2()
{
    lastVal = 23;
    ActivateTask(Task1); // System Call
}
void Task1()
{
    int temp = lastVal + 1000;
    data[0] = data[0] + temp;
}
void Task2()
{
    lastVal = 23;
    ActivateTask(Task1); // System Call
}
Compiler Optimization on System Level (potential)

```c
void Task1()
{
    int temp = lastVal + 1000;
    data[0] = data[0] + temp;
}
void Task2()
{
    lastVal = 23;
    ActivateTask(Task1); // System Call
}
```

Constant Propagation across Kernel Boundaries

```c
void Task1()
{
    data[0] = data[0] + 1023;
}
void Task2() { /* unchanged */}
```
A System Model for the Compiler

Problem: System-Calls are not transparent for the compiler
- Compilers stay only within the language level
- Possible operating-system decisions are not taken into account
A System Model for the Compiler

Problem: System-Calls are not transparent for the compiler
- Compilers stay only within the language level
- Possible operating-system decisions are not taken into account

Solution: We supply an OS execution model
- Knowledge about application–OS interaction
- Execution model includes possible scheduling decision
- System calls become more transparent for the compiler

Especially useful for embedded real-time systems
Application and kernel are often statically combined
Precise OS execution model through determinism
A System Model for the Compiler

Problem: System-Calls are not transparent for the compiler
- Compilers stay only within the language level
- Possible operating-system decisions are not taken into account

Solution: We supply an OS execution model
- Knowledge about application–OS interaction
- Execution model includes possible scheduling decision
- System calls become more transparent for the compiler

Especially useful for embedded real-time systems
- Application and kernel are often statically combined
- Precise OS execution model through determinism
Outline

- **Question 1:**
  How to gather OS execution model for a static real-time systems?

- **Question 2:**
  How to utilize the gathered information?
Outline

- **Question 1:**
  How to gather OS execution model for a static real-time systems?
  - Global Control-Flow Graph

- **Question 2:**
  How to utilize the gathered information?
Basic assumptions for our system-level analysis

- Event-triggered real-time systems: execution threads, interrupts, etc.
- Static system design: fixed number of threads, fixed priority
- Deterministic system-call semantic and scheduling
- System-calls are fixed in location and arguments
Basic assumptions for our system-level analysis

- Event-triggered real-time systems: execution threads, interrupts, etc.
- **Static system design**: fixed number of threads, fixed priority
- **Deterministic** system-call semantic and scheduling
- System-calls are fixed in location and arguments

Assumption apply to a wide range of systems: OSEK, AUTOSAR

- Industry standard widely employed in the automotive industry
- Static configuration at compile-time
Example Application

Static System Configuration

```plaintext
TASK TaskA {
    PRIORITY = 0;
    AUTOSTART = TRUE;
};

TASK TaskB {
    PRIORITY = 10;
};
```

Application Code

```plaintext
void TaskA() {
    int val = readData();
    buf.append(val);
    if (val != '\n') {
        buf.finalize();
        ActivateTask(TaskB);
        buf.clear();
    }
    TerminateTask();
}

void TaskB() {
    buf.print();
    TerminateTask();
}
```
Control-Flow Graph

TaskA (priority: 0)

val = readData();
buf.append(val);
if (val != '\n')

buf.finalize();
ActivateTask(TaskB)
buf.clear();
TerminateTask();

TaskB (priority: 10)

buf.print();
TerminateTask();
Global Control-Flow Graph (GCFG)

TaskA (priority: 0)

val = readData();
buf.append(val);
if (val != 'n')
buf.finalize();
ActivateTask(TaskB)
buf.clear();
TerminateTask();
buf.print();
TerminateTask();

TaskB (priority: 10)

buf.print();
TerminateTask();

GCFG

CFG

Computation
System Call
The GCFG contains all possible scheduling decisions
- GCFG is OS specific
- GCFG is application specific
GCFG and System State Enumeration

- The GCFG contains all possible scheduling decisions
  - GCFG is OS specific
  - GCFG is application specific

- Combine three information sources in System-State Enumeration
  - System specification
  - Static system configuration
  - Application structure from control-flow graphs
GCFG and System State Enumeration

- The GCFG contains all possible scheduling decisions
  - GCFG is OS specific
  - GCFG is application specific

- Combine three information sources in System-State Enumeration
  - System specification
  - Static system configuration
  - Application structure from control-flow graphs

- Basic principle of system-state enumeration
  - Instantiate abstract OS model with system configuration
  - Simulate the application structure on top of the OS model
  - Discover all possible system states
System-State Enumeration and the Transition Graph

State Transition Graph

TerminateTask()

Abstract System State

<table>
<thead>
<tr>
<th>Task State</th>
<th>TaskA</th>
<th>TaskB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ready</td>
<td></td>
<td>running</td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Resume Point</td>
<td>buf.clear();</td>
<td>TerminateTask()</td>
</tr>
</tbody>
</table>

Next Block

TaskB TerminateTask()
System-State Enumeration and the Transition Graph

State Transition Graph

TerminateTask() → buf.clear()

Abstract System State

<table>
<thead>
<tr>
<th>Task</th>
<th>Task State</th>
<th>Priority</th>
<th>Resume Point</th>
<th>Next Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskA</td>
<td>running</td>
<td>0</td>
<td>buf.clear()</td>
<td>TaskA</td>
</tr>
<tr>
<td>TaskB</td>
<td>suspended</td>
<td>10</td>
<td>–</td>
<td></td>
</tr>
</tbody>
</table>

FAU The Global Control-Flow Graph
System-State Enumeration and the Transition Graph

State Transition Graph

```
TerminateTask()
buf.clear()
TerminateTask()
```

Abstract System State

<table>
<thead>
<tr>
<th>Task</th>
<th>TaskA</th>
<th>TaskB</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>running</td>
<td>suspended</td>
</tr>
<tr>
<td>Priority</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Resume Point</td>
<td>TerminateTask()</td>
<td></td>
</tr>
<tr>
<td>Next Block</td>
<td>TaskA</td>
<td>TerminateTask()</td>
</tr>
</tbody>
</table>
System-State Enumeration and the Transition Graph

State Transition Graph

<table>
<thead>
<tr>
<th>Task</th>
<th>State</th>
<th>Priority</th>
<th>Resume Point</th>
<th>Next Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskA</td>
<td>suspended</td>
<td>0</td>
<td></td>
<td>Idle</td>
</tr>
<tr>
<td>TaskB</td>
<td>suspended</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Abstract System State
Transforming the State-Transition Graph
Transforming the State-Transition Graph

Group States by Next Block
Transforming the State-Transition Graph

Group States by Next Block
Outline

- **Question 1:**
  How to gather OS execution model for a static real-time systems?
  - Global Control-Flow Graph

- **Question 2:**
  How to utilize the gathered information?
Outline

Question 1:
How to gather OS execution model for a static real-time systems?
    → Global Control-Flow Graph

Question 2:
How to utilize the gathered information?
    → Specialized System Calls
    → Assertions on the System State
    → Kernel as a Statemachne
    → …
Control-Flow Graph

TaskA (priority: 0)

val = readData();
buf.append(val);
if (val != '\n')
    buf.finalize();

ActivateTask(TaskB)

buf.clear();

TerminateTask();

TaskB (priority: 10)

buf.print();

TerminateTask();

GCFG ➔ CFG  
Computation  System Call
Traditional Library Operating System

ActivateTask(TaskB)

- call

ActivateTask(Task task)

- set_ready(task);
  Task next = schedule();
  dispatch(next);

- call

set_ready(Task task)

- ...

Task schedule()

- ...

Dictate on generality: "One size fits all"

- One system-call implementation for all system-call sites
- System-call must be callable from anywhere
- Code reuse saves flash memory
Specialized System Calls

set_ready(TaskB);
dispatch_start(TaskB);

Specialize each system-call site:

- Strip out computation steps with predictable outcome
- Trade-off between run time and code size
- Outgoing edges in the GCFG are possible schedule() results.
Evaluation Scenario

- Evaluation System: \textit{dOSEK (dependable OSEK)}
  - Fault-tolerant OSEK implementation for IA-32
  - Generative Approach
Evaluation Scenario

- Evaluation System: dOSEK (dependable OSEK)
  - Fault-tolerant OSEK implementation for IA-32
  - Generative Approach

- Scenario: Quadrotor Flight Control
  - 11 tasks, 3 alarms, 1 ISR
  - 53 system-call sites
  - Execute system for 3 hyperperiods
Question 1:
How to gather OS execution model for a static real-time systems?
   - Global Control-Flow Graph

Question 2:
How to utilize the gathered information?
   - Specialized System Calls
   - Assertions on the System State
   - Kernel as a Statemachine
   - ...
Conclusion and Future Work

“With Great Knowledge comes Great Optimization Potential.”

— SpiderGCC

Fine-Grained Analysis of Event-Triggered, Static Real-Time Systems
- The Global Control-Flow Graph includes the application–OS interaction
- Additional static knowledge from the state-transition graph

Fine-Grained Tailoring of Application and Kernel
- Reduction of kernel runtime by $\sim 30\%$
- Monitoring of static system properties: $\sim 50\%$ smaller SDC rate

Further Applications
- Improve worst-case execution time analysis of whole applications
- Replace Kernel by a State Machine (→ OSPERT’15)

Source code available at https://github.com/danceos/dosek
System State Assertions

/* Enter Hook */

ActivateTask(TaskB)

/* Leave Hook */

(empty)

assert ready(TaskA)
assert suspended(TaskB)

/* Leave Hook */

assert ready(TaskA)
assert suspended(TaskB)

buf.print();

assert ready(TaskA)
assert ready(TaskB)

/* Enter Hook */

TerminateTask();

/* Enter Hook */

buf.print();
System State Assertions

TaskA

/* Enter Hook */
assert ready(TaskA)
assert suspended(TaskB)
ActivateTask(TaskB)
/* Leave Hook */

(empty)

TaskB

buf.print();

/* Enter Hook */
TerminateTask();
/* Enter Hook */
assert ready(TaskA)
assert suspended(TaskB)
ActivateTask(TaskB)
/* Leave Hook */

(buf.print();)

/* Enter Hook */
assert ready(TaskA)
assert ready(TaskB)
TerminateTask();

(empty)
System State Assertions

TaskA

/* Enter Hook */
assert ready(TaskA)
assert suspended(TaskB)
ActivateTask(TaskB)
/* Leave Hook */
assert ready(TaskA)
assert suspended(TaskB)

(empty)

TaskB

buf.print();

/* Enter Hook */
assert ready(TaskA)
assert ready(TaskB)
TerminateTask()
Fault Injection of System-State Assertions
Results with 748 Assertions

Unprotected dOSEK

- Absolute SDC Counts
  - Base: 1,391,51
  - ... with Assertions: 685,14

Protection dOSEK

- Absolute SDC Counts
  - Base: 0.15
  - ... with Assertions: 0.08

- Instructions per Syscall
  - Base: 68
  - ... with Assertions: 85

  - Base: 270
  - ... with Assertions: 293
Results with 748 Assertions

Unprotected dOSEK

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>... with Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute SDC Counts</td>
<td>1,391.51</td>
<td>685.14</td>
</tr>
<tr>
<td>Percentage</td>
<td>-51%</td>
<td></td>
</tr>
</tbody>
</table>

Protected dOSEK

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>... with Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute SDC Counts</td>
<td>0.15</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Instructions per Syscall

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>... with Assertions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instructions</td>
<td>68</td>
<td>85</td>
</tr>
<tr>
<td>Percentage</td>
<td>+25%</td>
<td></td>
</tr>
</tbody>
</table>

FAU The Global Control-Flow Graph
Results with 748 Assertions

Unprotected dOSEK

- Absolute SDC Counts
  - Base: 1,391.51
  - ... with Assertions: 685.14
  - Decrease: 51%

- Instructions per Syscall
  - Base: 68
  - ... with Assertions: 85
  - Increase: 25%

Protected dOSEK

- Absolute SDC Counts
  - Base: 0.15
  - ... with Assertions: 0.08
  - Decrease: 49%

- Instructions per Syscall
  - Base: 270
  - ... with Assertions: 293
  - Increase: 9%