Operating System Mechanism for Continuation-based Fine-grained Threads on Dedicated & Commodity Processors

Shigeru Kusakabe†, Satoshi Yamada†, Mitsuhiro Aono†, Masaaki Izumi†, Satoshi Amamiya†, Yoshinari Nomura‡, Hideo Taniguchi‡, and Makoto Amamiya †

† Kyushu University ‡ Okayama University
Outline

• Introduction
• Thread Model
• OS Issues on FUCE
• OS Issues on Commodity Processor
• Concluding remarks
Introduction

Multithreading: available on commodity platforms, derived from sequential model

Our approach

Model: dataflow
   – natural to asynchronous/concurrent execution

Focus: architectures, languages, operating systems

Platform: dedicated & commodity processor
Introduction - on dedicated platform

Fuce: dedicated to fine-grained multithreading

Benchmarks were user applications,
How about operating systems?

System calls with I/O request
- Multithreading with continuation,
- Handling external events without "interrupt"
- Delivered without controller such as APIC
Introduction - on commodity platform

Dataflow concept useful on commodity platforms?

- flexible scheduling to reduce overhead

Wrapped System Call
- buffer split-phase system call requests
- reduce context (mode) changes
- enhance locality of reference
Outline

• Introduction
• Thread Model
• OS Issues on FUCE
• OS Issues on Commodity Processor
• Concluding Remarks
Zero-Wait Thread

- Program graph: nodes / threads, edges / continuation relations.
- Thread: synch. counter & instruction sequence (incl. continuation)
- A continuation instruction specifies its succeeding thread code and context, and decrements the synchronization counter of the target.
- If the counter becomes to zero, the thread becomes ready to run, and runs to completion without suspension once started.
Thread and Instance

Instance frame memory

Function instance

Instruction memory

Execution unit

Execution unit

Function instance

Thread and Instance
Split-phase

Thread with wait

Split-phase style

thread

request

long-latency operation

result

wait()

request

long-latency operation

result

zero-wait thread

zero-wait thread
Outline

• Introduction
• Thread Model
• OS Issues on FUCE
• OS Issues on Commodity Processor
• Concluding Remarks
Fuce Processor

CMT (Chip multithreading) being developed at Kyushu University

Main memory

Thread Execution Unit

D-Cache

Load/Store Unit

I-Cache

Thread Execution Unit

Register file

Thread Activation Controller

I/O controller

Pre-load Unit

I-Cache

Multi-threaded processor architecture diagram.
Thread Activation Controller

Base-address: pointer to data area
lock bit: semaphore
sync-count: # waiting continuations
fan-in: value of fan-in to the thread
code-entry: pointer to thread code

Activation Control Memory

<table>
<thead>
<tr>
<th>Instance ID</th>
<th>Thread entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td></td>
</tr>
</tbody>
</table>

Select ACM entry

Ready Thread Queue

Instances

- lock-bit
- sync-count
- fan-in
- code-entry

Thread entries
Handling External Event

Interrupt-based sequential approach
- Current thread
- Device
- Handler
- Interrupt
- Suspend & resume (save & restore)

Continuation-based zero-wait thread approach
- Current thread
- Device
- Handler
- Trigger
- Executable in parallel

MTAAP / Mar. 30, 2007
Thread Mode

- User mode
- Supervisor mode

- User thread
- Kernel interface thread
- Kernel thread

Not allowed
Critical Thread

Thread A
try to lock thread D

Thread B
try to lock thread D
succeed

Thread C
try to lock thread D
retry

Thread D
unlock myself

: continuation to another thread
: continuation to itself
Handling System calls with I/O Request

User

sender_threads

1-1: try to lock gate_thread
if (lock)
1-2: continue to the gate_thread
else
1-3: self-continuation

receiver_threads

7-1: receive data

Kernel

gate_thread

2-1: identify the requested system call ID
2-2: continue to the thread of the system call ID

syscall_thread

3-1: execute the body of system call
3-2: continue to the semaphore_thread

handler_thread

6-1: receive data
6-2: continue to the receiver thread with the result
if (queue is not empty)
6-3: extract data from queue and continue to device_thread
else
6-4 unlock device_thread

Device

semaphore_thread

4-1: try to lock the device_thread
If (lock)
4-2: continue to the device_thread
else
4-3: buffer data for I/O

device_thread

queue

5-1: receive data
5-2: issue I/O request
5-3: pass the receiver ID to handler_thread

: thread    : continuation    : data
Thread Activation

User

gate_thread

syscall_thread

Kernel

semaphore_thread

device_thread

Device

sender_thread

:executable in parallel
Measurement

Fuce in VHDL on ModelSim

Measured the number of system calls with I/O request ideally completed within a fixed period.

The number of TEUs: 1..4, devices: 1..3

Expectation: scalability -- activation of hander thread due to continuation mechanism
Evaluation Result

RTT: $\tau$ micro seconds

Scalability with \# TEUs

Scalability with \# devices

\# devices

period: 100 micro seconds
Outline

• Introduction
• Thread Model
• OS Issues on FUCE
• OS Issues on Commodity Processor
• Concluding Remarks
CEFOS: Process / Threads

Unix-like process, and dataflow-like thread
- dependence graph (partially ordered threads)
- process as thread context (color / tag)
CEFOS: Process / Threads

- Controls between threads: in a dataflow-like fashion
  - synchronization counter
  - serial code
  - continuation

Dataflow concept useful?
Preliminary experiment

LMbench result for Linux 2.6.14 - Latency benchmark (in clocks)

<table>
<thead>
<tr>
<th>processor</th>
<th>null call</th>
<th>2p/0K</th>
<th>2p/16K</th>
<th>L1</th>
<th>L2</th>
<th>M. Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>pentium III</td>
<td>378</td>
<td>1576</td>
<td>5044</td>
<td>3</td>
<td>8</td>
<td>164</td>
</tr>
<tr>
<td>pentium4</td>
<td>1090</td>
<td>3298</td>
<td>5798</td>
<td>2</td>
<td>18</td>
<td>261</td>
</tr>
<tr>
<td>PowerPC G4</td>
<td>200</td>
<td>788</td>
<td>2167</td>
<td>4</td>
<td>10</td>
<td>127</td>
</tr>
<tr>
<td>PowerPC G5</td>
<td>306</td>
<td>13698</td>
<td>13734</td>
<td>3</td>
<td>11</td>
<td>113</td>
</tr>
<tr>
<td>Intel Core Duo</td>
<td>464</td>
<td>1327</td>
<td>2820</td>
<td>3</td>
<td>14</td>
<td>152</td>
</tr>
</tbody>
</table>

System Call Overhead | Process Switch | Memory Latency
CEFOS - wrapped system-call

Partially ordered fine-grain threads
  split-phase style system calls
  … various choices in scheduling threads/processes

Wrapped System-Call (WSC)
  – aggregates multiple system-call requests
  – handles them as a single system-call
  to reduce overhead of system calls and enhance locality
CEFOS and WSC mechanism

- process
  - thread
  - thread
  - thread

- process
  - thread
  - thread
  - thread

- process
  - thread
  - thread
  - thread

Display Requests and Data

external kernel

internal kernel

user mode

supervisor mode
DRD

DRD (Display Request & Data)

Intermediate communications between Internal / External Kernel.

– Each process & kernel share common memory area (CA)
– Each process & kernel display requests and necessary information on CA
– At appropriate occasions, each process & kernel check requests and information displayed on CA, and change the control of execution if necessary.
Control flows in WSC

- **Process**
  - Thread
  - System-call

- **External kernel**
  - Buffer system-call request
  - 
    - # requests >= threshold ?
  - No
    - Thread scheduler
  - Yes
    - **Internal kernel**
      - Perform each system-call
      - User mode
      - Supervisor mode

- Return results & activate waiting threads
Impact on System Call Overhead

Implemented by modifying Linux.
Issuing system calls with thin body: getpid()
Locality of reference

- chatroom benchmark
  - simulate chat rooms (server and clients)
  - four threads per client (2 message handler (send /receive) in client & server)
  - parameters: number of clients = 20

Detailed memory events with performance monitoring counter - hardmeter (limited to focused part only)

<table>
<thead>
<tr>
<th></th>
<th>clocks</th>
<th>L2$ miss (%)</th>
<th>D-TLB miss (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>normal</td>
<td>60217</td>
<td>1.01</td>
<td>2.78</td>
</tr>
<tr>
<td>WSC</td>
<td>48436</td>
<td>0.47</td>
<td>2.55</td>
</tr>
<tr>
<td>ratio: WSC/normal</td>
<td>0.80</td>
<td>0.47</td>
<td>0.92</td>
</tr>
</tbody>
</table>
Concluding remarks
Multithreading based on dataflow model

On Fuce
• event handling without “interrupt”

On commodity platforms
• Wrapped System-Call: aggregates split-phase style system call requests

Evaluation
– scalability of throughput in handling I/O request
– system call overhead and locality of reference