Delphi Tools Update: Instrumenting Threaded Programs

Barton P. Miller
bart@cs.wisc.edu

Computer Science Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706-1685
USA
Outline

• Motivation
• Design Issues
• Implementation Highlights
• Current Status
• Future Work
Motivation

• Why support multithreaded applications?
  – Exploit multiprocessor hardware, application concurrency
  – Used heavily in transaction processing, UI’s, servers
Motivation

• Metric computation. What is new?
  – For single or few threads:
    “cpu time for thread 1 / process 2”
    “cpu time for thread 2 / process 2”
  – For all threads, individually
  – For all threads, cumulative
Previous Paradyn Program Instrumentation

Process 1

Instr. code

Counter 1
Counter 2
Timer 1
Counter 3
Timer 2
Timer 3
Counter 4

Process 2

Instr. code

Counter 1
Counter 2
Timer 1
Counter 3
Timer 2
Timer 3
Counter 4

One process, one thread

Vector of CT1

Vector of CT2

# Program Instrumentation w/Threads

## One process, one thread

<table>
<thead>
<tr>
<th>Process 1</th>
<th>Vector of CT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr. code</td>
<td></td>
</tr>
<tr>
<td>counter 1</td>
<td></td>
</tr>
<tr>
<td>counter 2</td>
<td></td>
</tr>
<tr>
<td>timer 1</td>
<td></td>
</tr>
<tr>
<td>counter 3</td>
<td></td>
</tr>
<tr>
<td>timer 2</td>
<td></td>
</tr>
<tr>
<td>timer 3</td>
<td></td>
</tr>
<tr>
<td>counter 4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector of CT2</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter 1</td>
</tr>
<tr>
<td>counter 2</td>
</tr>
<tr>
<td>timer 1</td>
</tr>
<tr>
<td>counter 3</td>
</tr>
<tr>
<td>timer 2</td>
</tr>
<tr>
<td>timer 3</td>
</tr>
<tr>
<td>counter 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process 2</th>
<th>Instr. code</th>
</tr>
</thead>
</table>

## One process, multiple threads

<table>
<thead>
<tr>
<th>Process</th>
<th>Vector of CT Thread 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
</tr>
<tr>
<td>counter 1</td>
<td></td>
</tr>
<tr>
<td>counter 2</td>
<td></td>
</tr>
<tr>
<td>timer 1</td>
<td></td>
</tr>
<tr>
<td>counter 3</td>
<td></td>
</tr>
<tr>
<td>timer 2</td>
<td></td>
</tr>
<tr>
<td>timer 3</td>
<td></td>
</tr>
<tr>
<td>counter 4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vector of CT Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>counter 1</td>
</tr>
<tr>
<td>counter 2</td>
</tr>
<tr>
<td>timer 1</td>
</tr>
<tr>
<td>counter 3</td>
</tr>
<tr>
<td>timer 2</td>
</tr>
<tr>
<td>timer 3</td>
</tr>
<tr>
<td>counter 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>One process, multiple threads</th>
<th>Instr. code</th>
</tr>
</thead>
</table>
Design issues

• Every thread shares the same program instrumentation

• Vector of counters or timers per thread
  – More memory usage, but better speed
  – More straight forward implementation

• Two base applications scenarios
  – Few threads, few LWPs: exploiting parallelism
  – Many threads, dynamic (e.g. servers)
### Design for Instrumenting Threads

#### ThreadTable

<table>
<thead>
<tr>
<th>Metric</th>
<th>Thr1</th>
<th>Thr2</th>
<th>ThrN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metric 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metric 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Vector of Counters

Base Address

Offset for C4

#### Vector of Timers

- T1
- T2
- Tn
- C1
- C2
- C3
- C4
- Cn
Design for Instrumenting Threads

• Whole process vs. Threads
  – Important performance issue!
  – Whole process metrics are computed per process ⇒ no need to aggregate threads
  – “thr_1” is equivalent to whole process
  – Aggregation is done for processes, not for threads
Original Base Trampoline

Application Program

Func foo:

Base Trampoline

Save Regs
Update cost
Pre instrument.
Restore Regs
Relocated Instruction(s)
Save Regs
Update cost
Post instrument.
Restore Regs

Mini Trampoline

Instrumentation Primitive (e.g. addCounter)
Modified Base Trampoline

```
Func foo:
```

Application Program

Base Trampoline

Save Regs
Update cost
MTPreamble
Pre instrument.
Restore Regs
Relocated Instruction(s)
Save Regs
Update cost
MT Preamble
Post instrument.
Restore Regs

Mini Trampoline

Compute C/T Addr. (per thread)
Instrumentation Primitive (e.g. addCounter)
Current Design - Issues

- Thread Table indexed by thread id’s, points to vector of counters or timers
- Separate sets of counters/timers per thread
- Creation of vectors of counters/timers on demand, never removed, but re-used!
- Counters/timers allocated by blocks
- Virtual CPU timer for each thread
Current Design - Key Operations

• Add thread
  – Update thread table entry
  – Create same counter/timers as for other threads
  – Enable only counter/timers that apply to new thread

• Delete thread
  – De-allocate all counter/timers + all vectors for this thread
  – Update thread table entry
Current Design - Key Operations

- Add counter/timer
  - Common case: there is space in vector of counter/timers and we just add new entry
  - Special case: there is no space available and we create a new vector for all threads and add new entry

- Delete counter/timer
  - Tag counter/timer as invalid. It does not de-allocate memory
Current Design - Comments

• Advantages
  – Reasonable memory usage
  – Fast execution of mini-trampoline code

• Disadvantage
  – Only de-allocates memory for counter/timers and vectors when a thread is deleted
Example Measurements
Example Measurements

![Graph showing example measurements with various lines representing different metrics over time.](image)
Current Design - Comments

• Must instrument thread context switch
  – Identify appropriate functions in thread package

• For Solaris threads
  – "_onproc_deq": stop timer, thr context switch
  – "_resume_ret": start timer, thread is about to resume execution

• A little messy: requires internal knowledge
  – Only done once per thread package
Current Design - CPU metric

resourceList stopThread is procedure {
    items {"_onproc_deq"};
    flavor { unix };
    library true;
}

resourceList resumeThread is procedure {
    items {"_resume_ret"};
    flavor { unix };
    library true;
}
Current Design - CPU metric

metric cpuTime {
...
base is processTimer {
    foreach func in stopThread {
        append preInsn func.entry (* stopProcessTimer(cpuTime); *)
    }
    foreach func in resumeThread {
        append preInsn func.entry (* startProcessTimer(cpuTime); *)
    }
    append preInsn $start.entry constrained (* startProcessTimer(cpuTime); *)
    prepend preInsn $start.return constrained (* stopProcessTimer(cpuTime); *)
    append preInsn $exit.entry constrained (* stopProcessTimer(cpuTime); *)
}
}
Current Design - Cost

Base-Trampoline Section. MT Version (SPARC Architecture)

// Instrumentation code - Part of the Base Trampoline
// MT Preamble
basetramp:  sethi  %hi(0x12400), %o5
basetramp+4:  call  %o5 + 0x3dc ! 0x127dc<DYNINSTthreadPos>
basetramp+8:  nop
basetramp+12:  sll  %o0, 2, %l0
basetramp+16:  sethi  %hi(0x42b400), %l1
basetramp+20:  or  %l1, 0x130, %l1
basetramp+24:  add  %l0, %l1, %l0
basetramp+28:  mov  %l0, %l7
basetramp+32:  nop
Current Design - Cost

Mini-trampoline (SPARC architecture)

// Instrumentation code ("add counter" primitive)
// Load counter
minitramp: sethi %hi(0x61800),%l0
minitramp+4: ld [%l0+0x3e0],%l0 ! 0x61be0
<DYNINSTdata+1760>
// Increment counter
minitramp+8: inc %l0
// Store counter
minitramp+12: sethi %hi(0x61800),%l1
minitramp+16: st %l0, [%l1+0x3e0] ! 0x61be0
<DYNINSTdata+1760>
// Branch to base trampoline or next mini-trampoline
minitramp+20: b,a basetramp
minitramp+24: nop
Current Design - Cost

Mini-trampoline. MT Version (SPARC Architecture)

// Instrumentation code ("add counter" primitive)
// Load CT Vector Address
minitramp:  ld  [ %17 ], %12
// Compute offset for this counter
minitramp+4:  mov  0x12b, %13
minitramp+8:  sll  %13, 0x42, %13
minitramp+12:  add  %12, %13, %12
// Load counter address and value
minitramp+16:  ld  [ %12 ], %11
minitramp+20:  ld  [ %11 ], %10
// Increment and store counter
minitramp+24:  inc  %10
minitramp+28:  st  %10, [ %11 ]
// Branch to base trampoline or next mini-trampoline
minitramp+32:  b,a  basetramp
minitramp+36:  nop
Current Status

• Solaris threads support

• Thread low-level instrumentation in place and working

• Measurements can be gathered for a multiple threads using new structure
What is next?

- Testing small multithreaded applications running on multiprocessors
  - Exploit relationship threads/LWPs/CPUs
- Evaluate and tune performance
- Test large-scale application: Oracle on Solaris is initial target.