

University of
Waterloo



SMP Concurrent Software Development in C++

Peter A. Buhr (faculty)
Richard Bilson (staff)
Ashif Harji (Ph.D.)
Jiongxiang Chen (Ph.D.)
Roy Krischer (Ph.D.)
Josh Lessard (M.Math)

May 25, 2004



Outline

1. $\mu\text{C++}$: concurrency in C++
2. $\mu\text{Profiler}$: profiling $\mu\text{C++}$ applications
3. Threaded web-server in $\mu\text{C++}$

μ C++ : Concurrency in C++

- C++ has no concurrency
- μ C++: translator/library providing integrated object-oriented concurrency
- uses an M:N thread model (versus 1:1 NPPTL)
- two new type constructors:
 - **coroutine** / **task**, extensions of **class**
 - provide stack and thread
- two new type qualifiers:
 - **mutex** / **nomutex**, qualify type constructors and routines
 - provide mutual exclusion
- 6 new statements:
 - **suspend** / **resume** (context switch)
 - **wait** / **signal** / **signalblock** / **accept** (synchronization)

MUTUAL EXCLUSION

<pre>class c { public: m() { } };</pre>	<pre>coroutine C { main() { suspend } public: m() { resume } };</pre>	<pre>mutex class M { condition variables public: m() { wait/signal/accept } };</pre>
<p>STACK</p>	<pre>mutex coroutine CM { condition variables main() { suspend } public: m1() { resume } m2() { wait/signal/accept } };</pre>	<pre>task T { condition variables main() { suspend/wait/signal/accept } public: m1() { resume } m2() { wait/signal/accept } };</pre>
<p>STACK / THREAD</p>		

MONITOR

TASK

<pre> mutex class ReadersWriter { int rcnt, wcnt; public: void ReadersWriter() { rcnt = wcnt = 0; } void StartRead() { if (wcnt > 0) accept(EndWrite); rcnt += 1; } void EndRead() { rcnt -= 1; } void StartWrite() { if (wcnt > 0) accept(EndWrite); else while (rcnt > 0) accept(EndRead); wcnt = 1; } void EndWrite() { wcnt = 0; } }; </pre>	<pre> task Server { condition delay; void main() { for (;;) { accept(~Server) { break; } or accept(workReq1) { // process work } or accept(workReq2) { ... } } // shut down } public: void workReq1(Req1 req) { ... wait delay; ... } void workReq2(Req2 req) { ... } ... }; </pre>
---	--

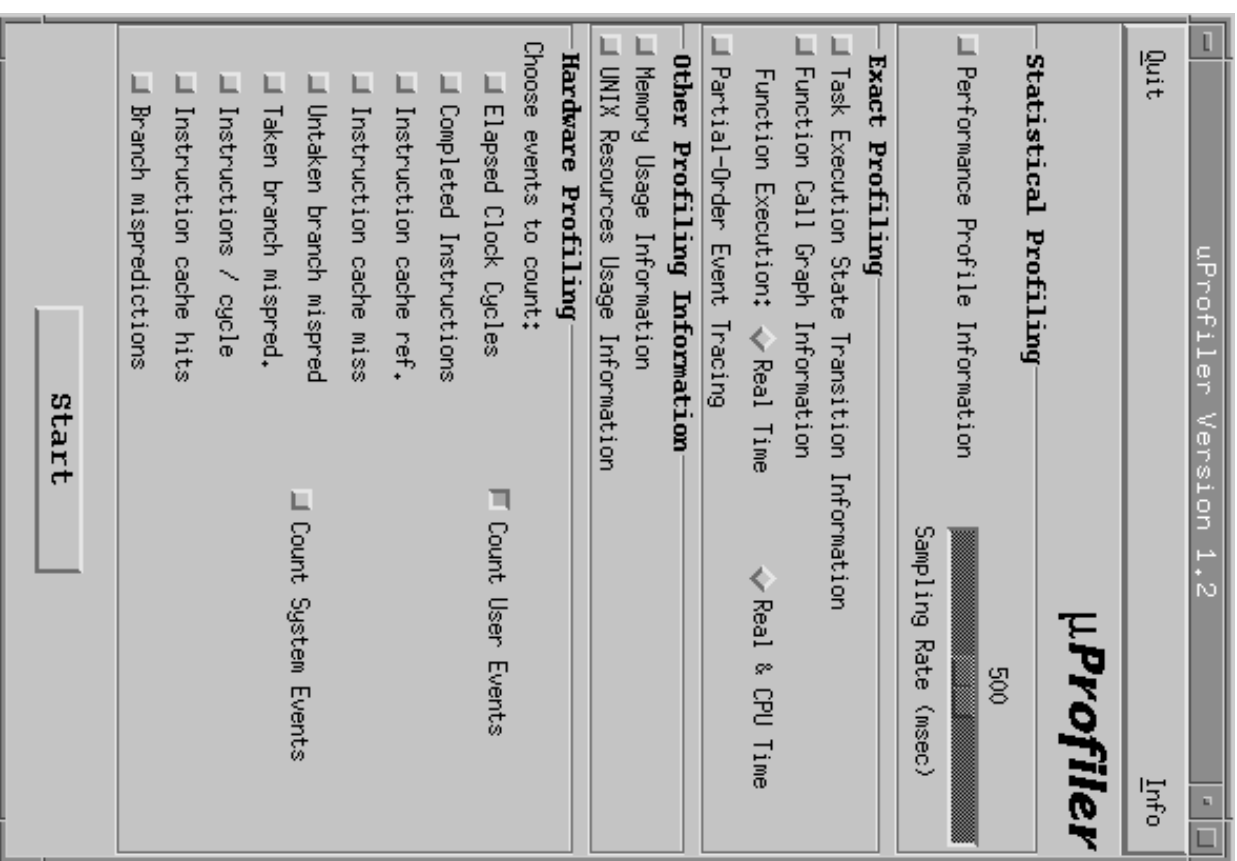
- real-time support
 - extensible schedulers (fixed, dynamic)
 - transitive inheritance protocol
 - timeout facility
 - accept** (M1, M2) { ... }
 - or accept** (M3) { ... }
 - or timeout**(1) { ... } // after 1 second
 - real-time tasks:
 - periodic** P { ... };
 - aperiodic** A { ... };
 - sporadic** S { ... };
- polymorphism: inheritance and templates with new types
- user-defined grouping of tasks and *virtual* processors (cluster)
- advanced concurrent exception handling
 - throw** [*throwable-event*] [**at** *coroutine/task-id*]] ; // termination
 - raise** [*resumable-event*] [**at** *coroutine/task-id*]] ; // resumption

- object-oriented, nonblocking I/O for files and sockets
 - classes for socket server, acceptor and client
 - simplified programming interface
- debug mode for testing (asserts and runtime checks)
- ports: gcc-3.X for **Linux IA-64 (HP/SGI)**, Linux IA-32/AMD, Solaris SPARC, IRIX MIPS
- future work:
 - increase performance
 - scalability testing (NUMA)
 - port to Intel compiler (**asm** statements?)
- <http://plg.uwaterloo.ca/~usystem/uC++.html>

μ Profiler: Profiling in μ C++

- provides information about dynamic execution
- integrated with μ C++ programming model
 - information provided per-task, per-coroutine, per-routine
 - trace multiple concurrent events
 - more effective, efficient and extensible profiling
- design:
 - concurrent library written in μ C++
 - executes within μ C++ application
 - extensible: user-definable
 - * metrics (data gathering)
 - * displays (data visualization)
- uses concurrent version of X/Motif for display

- compile with -profiler flag
- run application
- choose metrics:
 - statistical:
 - * execution time
 - exact:
 - * state transition
 - * dynamic call-graph, execution time, hardware counters
 - * partial-order tracing
 - * memory leaks
 - * kernel-threads
- press **Start**



Statistical: Execution Time

Performance Profiling Analysis - Top Level

Close

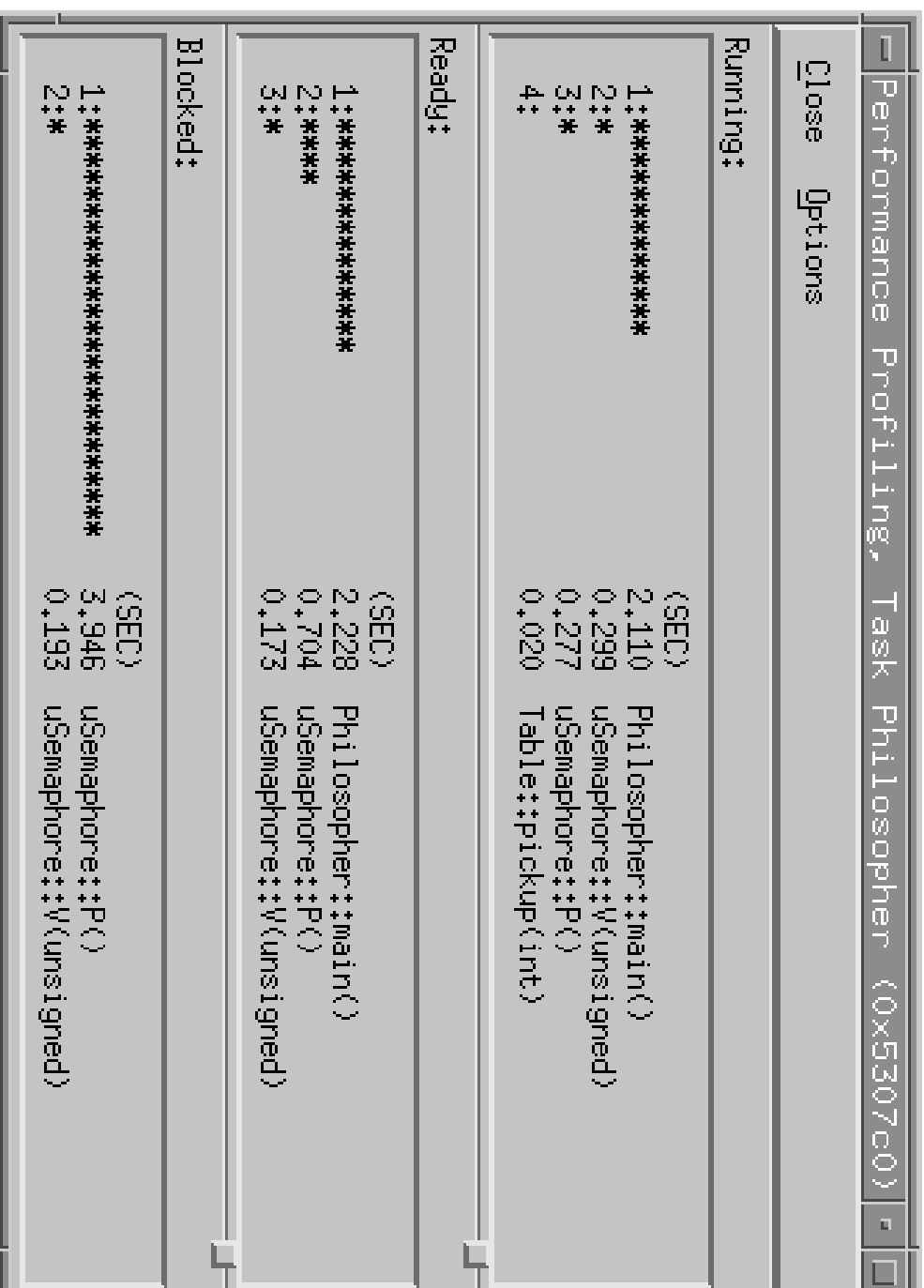
Cluster Name (ID)	Tasks	Max Running Time (sec)	Active Time (%)
UserCluster (0x155370)	7	2,890	21.9
uSystemCluster (0x108a20)	1	0,000	0,0

Minimum Sampling Frequency: 12 ms
 Maximum Sampling Frequency: 87 ms
 Number of Samples: 205

Performance Profiling, Cluster UserCluster (0x155370)

Close

Task Name (ID)	Running (sec)	Ready (sec)	Blocked (sec)	Undetermined (sec)	Life Time (sec)	Coverage (%)
Philosopher (0x52fa80)	2,890	2,776	4,287	0,000	9,953	100,0
Philosopher (0x4e2c00)	2,738	3,425	3,790	0,000	9,953	100,0
Philosopher (0x5307c0)	2,707	3,106	4,140	0,000	9,953	100,0
Philosopher (0x4dc510)	2,442	2,813	4,699	0,000	9,954	100,0
Philosopher (0x4ec7f0)	2,325	3,223	4,405	0,000	9,953	100,0
uMain (0xffbef838)	0,000	0,000	9,954	0,000	9,954	100,0
uBootTask (0x108a60)	0,000	0,000	0,000	0,000	0,000	0,0



Exact: State Transition

Execution State Transition Information: Task Selection

Close Options

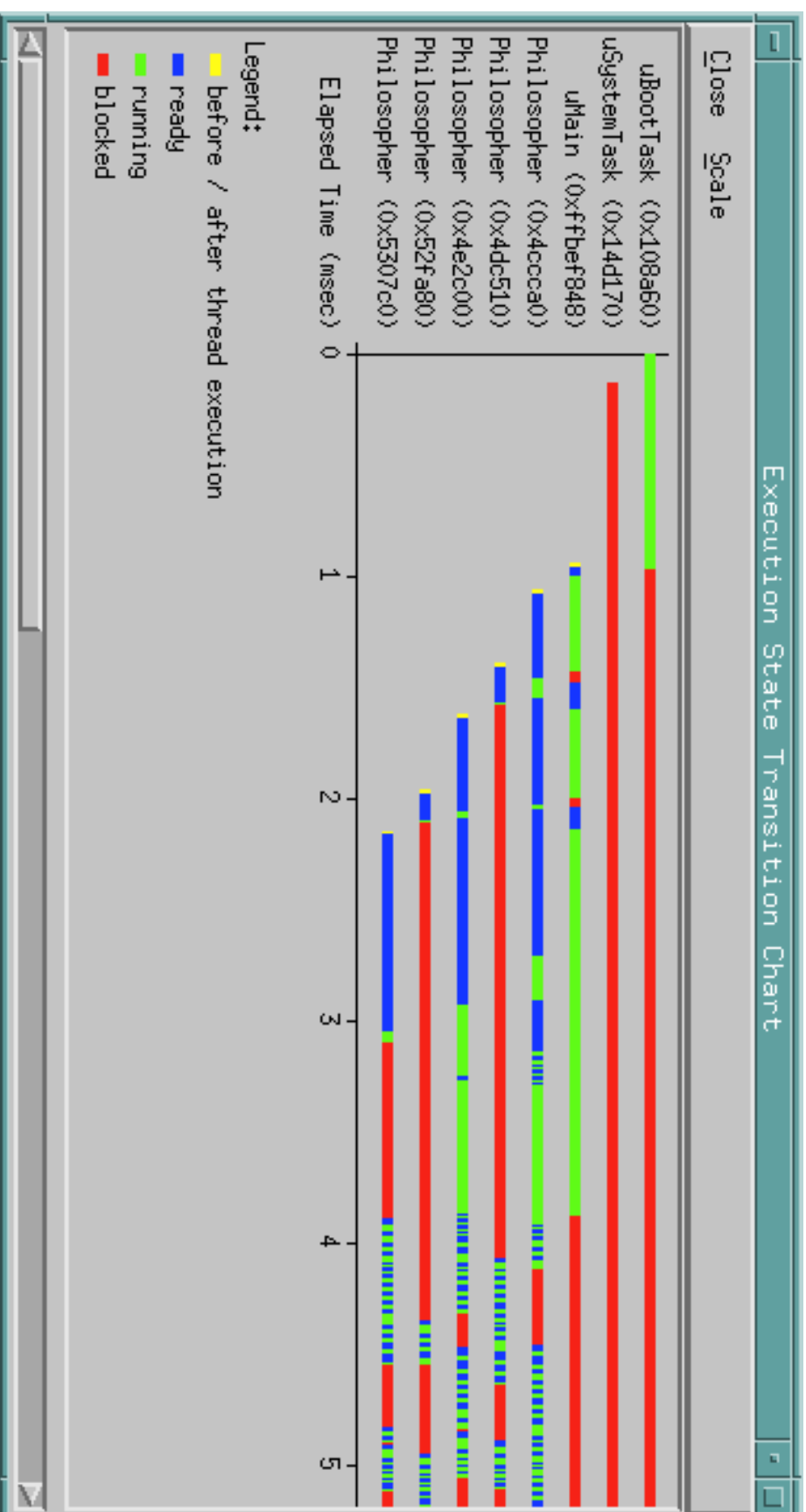
Task Name (ID)	Creation Time (h:m:s.ms)	Created	Deleted	Number of Tasks Started main()	Ended main()	Executing
uBootTask (0x108a60)	11:08:18.815	1	0	1	0	1
uSystemTask (0x14d170)	11:08:18.815	2	0	2	1	1
uMain (0xffbef848)	11:08:18.815	3	0	2	1	2
Philosopher (0x4ccca0)	11:08:18.815	4	0	3	1	3
Philosopher (0x4dc510)	11:08:18.816	5	0	3	1	4
Philosopher (0x4e2c00)	11:08:18.816	6	0	5	1	5
Philosopher (0x52fa80)	11:08:18.816	7	0	5	1	6
Philosopher (0x5307c0)	11:08:18.816	8	0	7	1	7

Task Creation Graph

Close

```

uBootTask (0x108a60)
  uSystemTask (0x14d170)
    uMain (0xffbef848)
      Philosopher (0x4ccca0)
      Philosopher (0x4dc510)
      Philosopher (0x4e2c00)
      Philosopher (0x52fa80)
      Philosopher (0x5307c0)
    
```



Task Philosopher (0x4ccca0): Execution State Transition

Close

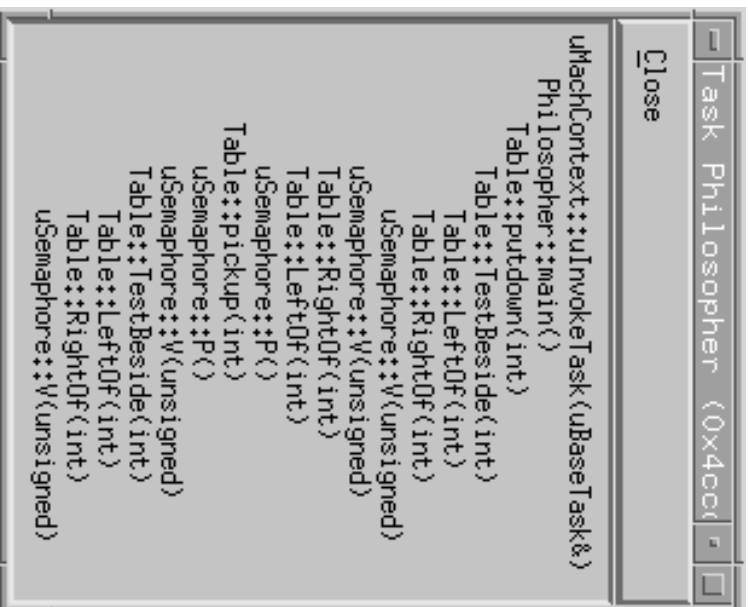
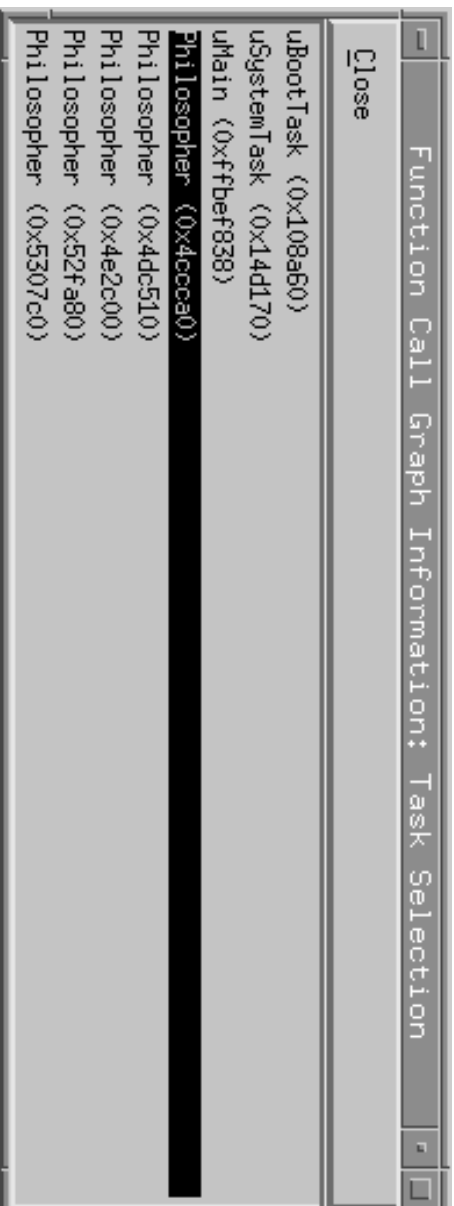
Execution Summary

Life Time:	(MSEC)	(%)	Clock Time:	(H:M:S,MS,US)
ready	11,093	100,00	Creation	11:08:18,815,973
running	4,462	40,22	Deletion	11:08:18,827,066
blocked	3,934	35,46		
State Duration:	2,630	23,71		
Minimum	0,008			
Maximum	0,774			

State Transitions

No.	State	Duration (msec)	Cum. Duration (msec)	Start Duration (msec)	Started in Function
1	start	0,018	0,018	0,965	*unknown*
2	ready	0,347	0,365	1,312	*unknown*
3	running	0,075	0,440	1,387	*unknown*
4	ready	0,424	0,864	1,811	Philosopher::main()
5	running	0,016	0,880	1,827	Philosopher::main()
6	ready	0,090	0,970	1,917	Philosopher::main()
7	running	0,014	0,984	1,931	Philosopher::main()
8	ready	0,774	1,758	2,705	Philosopher::main()
9	running	0,021	1,779	2,726	Philosopher::main()
10	ready	0,097	1,876	2,823	Philosopher::main()
11	running	0,164	2,040	2,987	Philosopher::main()
12	ready	0,013	2,053	3,000	Philosopher::main()
13	running	0,017	2,070	3,017	Philosopher::main()

Exact: Call Graph



Task Philosopher (0x4b9868) : Function Call Information

Close Options

Call Cycles:

<no call cycles detected>

Function Real and CPU Times:

From/To	Calls	Average	REAL TIMES (msec)		Total	Average	CPU TIMES (msec)		Total
			Minimum	Maximum			Minimum	Maximum	
Table::TestBeside(int)	49779	0.008	0.004	0.051	440.916	0.005	0.002	0.020	284.044
Table::LeftOf(int)	49779	0.001	0.001	0.002	73.127	0.001	0.001	0.002	73.127
Table::RightOf(int)	26241	0.001	0.001	0.027	40.483	0.001	0.001	0.027	40.483
uSemaphore::V(unsigned)	16596	0.002	0.002	0.016	43.262	0.002	0.002	0.016	43.262
Table::pickup(int)	16593	0.550	0.028	5.463	9139.493	0.010	0.009	0.022	181.759
uSemaphore::P()	33186	0.264	0.001	5.443	8784.725	0.012	0.001	0.097	410.101
uSemaphore::V(unsigned)	16593	0.002	0.002	0.012	36.080	0.002	0.002	0.012	36.080
Table::TestBeside(int)	16593	0.008	0.004	0.051	136.929	0.005	0.003	0.020	90.803
Table::putdown(int)	16593	0.041	0.031	0.064	688.667	0.015	0.012	0.035	256.878
Table::TestBeside(int)	33186	0.009	0.004	0.028	303.987	0.005	0.002	0.016	193.241
uSemaphore::V(unsigned)	16593	0.002	0.001	0.003	35.330	0.002	0.001	0.003	35.330
Table::RightOf(int)	16593	0.001	0.001	0.005	25.525	0.001	0.001	0.005	25.525
Table::LeftOf(int)	16593	0.001	0.001	0.002	25.595	0.001	0.001	0.002	25.595
uSemaphore::P()	16593	0.002	0.002	0.003	41.352	0.002	0.002	0.003	41.352
Philosopher::main()	1	26395.889	26395.889	26395.889	26395.889	3817.421	3817.421	3817.421	3817.421
Table::putdown(int)	16593	0.041	0.031	0.064	688.667	0.015	0.012	0.035	256.878
Table::pickup(int)	16593	0.550	0.028	5.463	9139.493	0.010	0.009	0.022	181.759
uHackContext::uInvokeTask(uBaseTask&)	1	26395.889	26395.889	26395.889	26395.889	3817.421	3817.421	3817.421	3817.421
Philosopher::main()	1	26395.889	26395.889	26395.889	26395.889	3817.421	3817.421	3817.421	3817.421

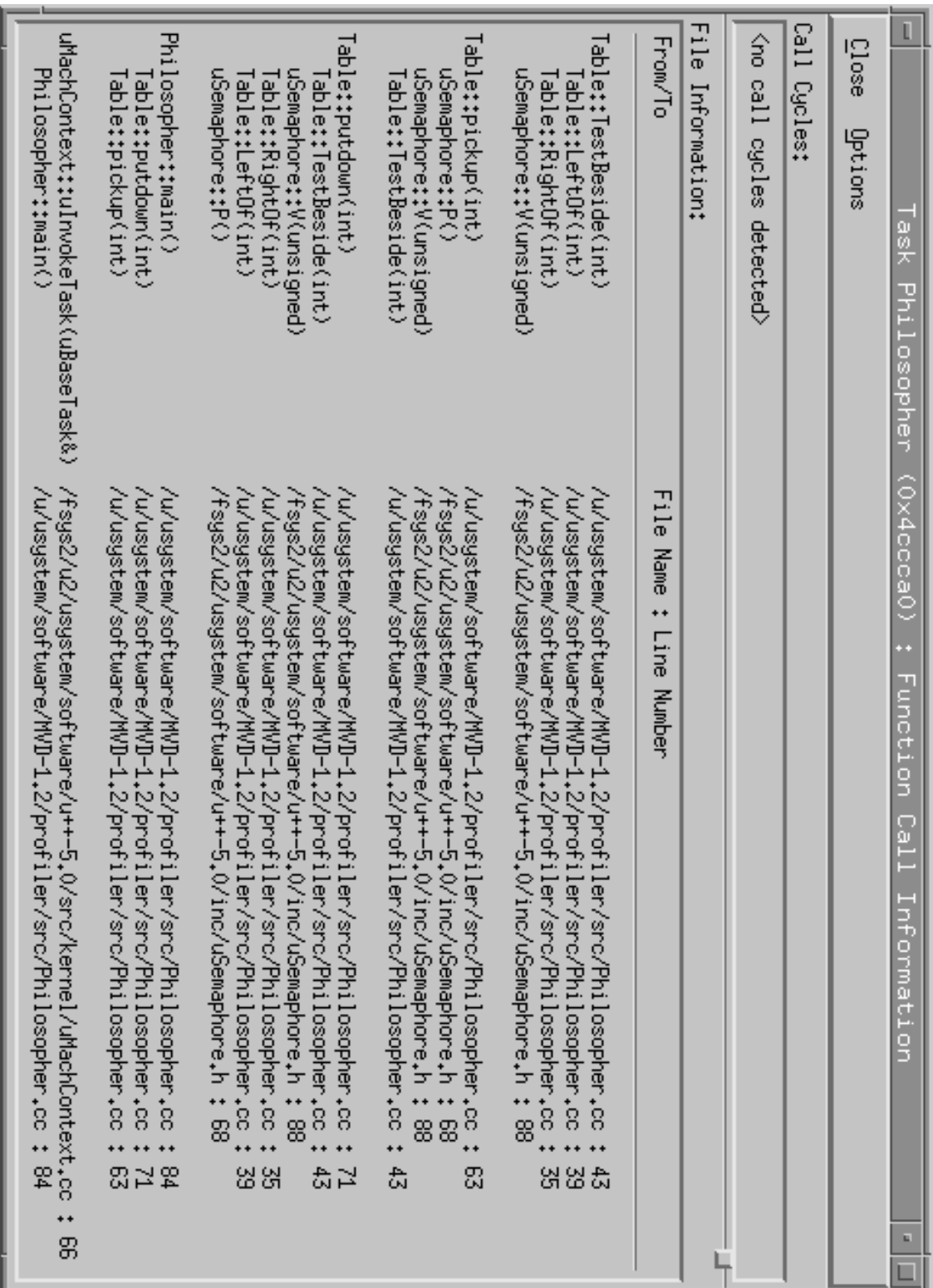
Task Philosopher (0x4b9868) : Function Call Information

Close Options

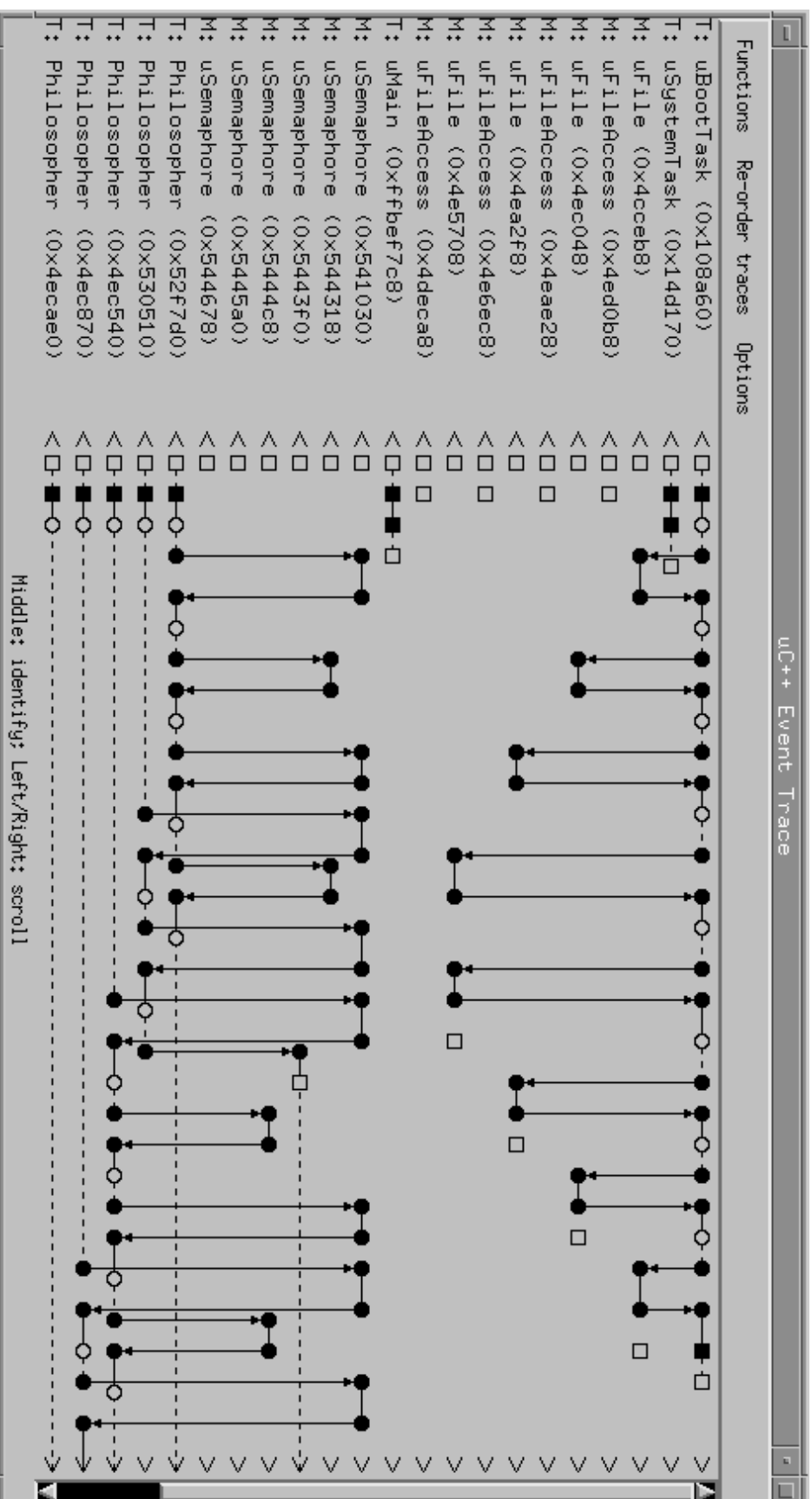
Call Cycles:
<no call cycles detected>

Function Event Counts:

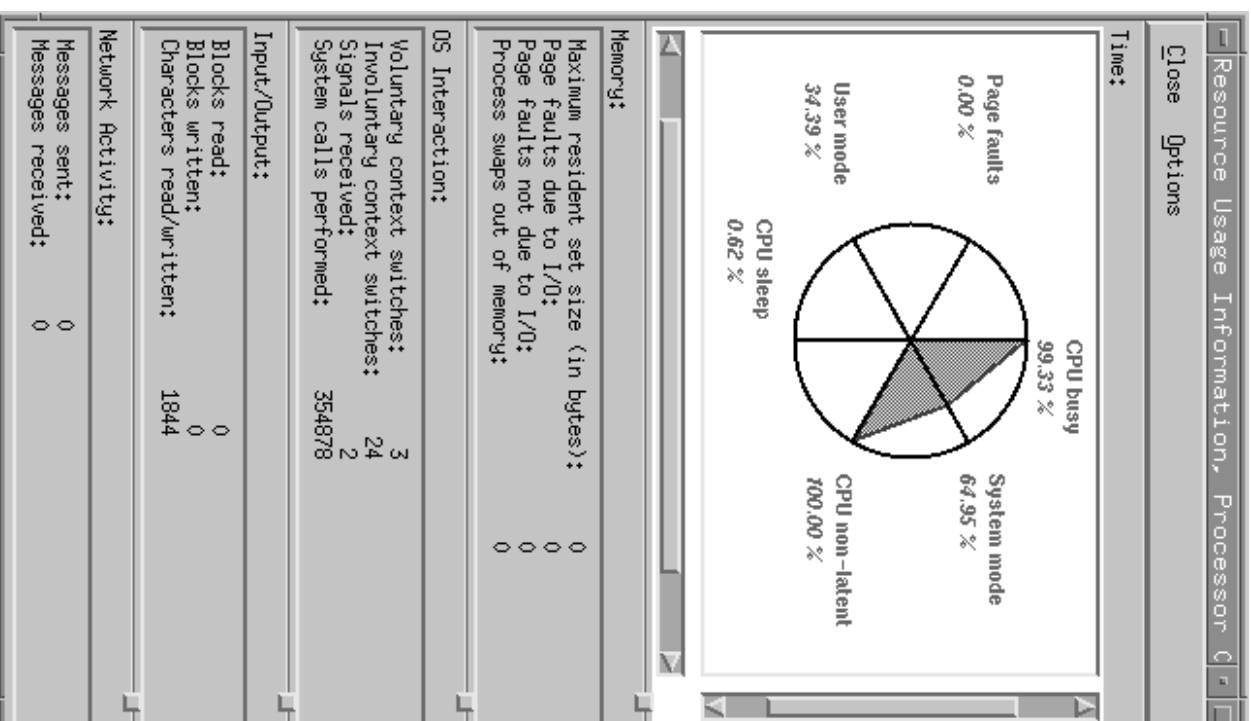
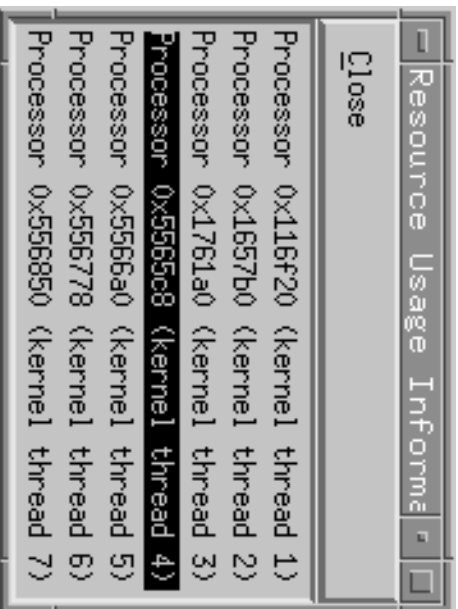
From/To	Calls	Completed Instructions		Instruction cache miss	
		Average	Total	Average	Total
Table::TestBeside(int)	49779	2079,197	103500348	11,586	576751
Table::LeftOf(int)	49779	872,714	43442844	4,522	225084
Table::RightOf(int)	26241	885,691	23241408	4,613	121050
uSemaphore::W(unsigned)	16593	1356,991	22520619	35,674	592054
Table::pickup(int)	16593	3453,759	57308227	25,036	415428
uSemaphore::P()	33186	1470,012	48783826	58,239	1932711
uSemaphore::W(unsigned)	16593	1323,529	21961318	13,783	228704
Table::TestBeside(int)	16593	1988,481	32994866	13,564	225063
Table::putdown(int)	16593	4952,197	82171809	25,709	426596
Table::TestBeside(int)	33186	2124,555	70505482	10,597	351688
uSemaphore::W(unsigned)	16593	1323,384	21958906	11,232	186375
Table::RightOf(int)	16593	886,135	14703642	3,961	65726
Table::LeftOf(int)	16593	870,781	14448861	6,689	110984
uSemaphore::P()	16593	1297,075	21522365	51,228	850020
Philosopher::main()	1	56481111,000	56481111	3110770,000	3110770
Table::putdown(int)	16593	4952,197	82171809	25,709	426596
Table::pickup(int)	16593	3453,759	57308227	25,036	415428
uMachContext::uInvokeTask(uBaseTask&)	1	56481111,000	56481111	3110770,000	3110770
Philosopher::main()	1	56481111,000	56481111	3110770,000	3110770



Exact: Partial-Order Tracing



Exact: Kernel Threads

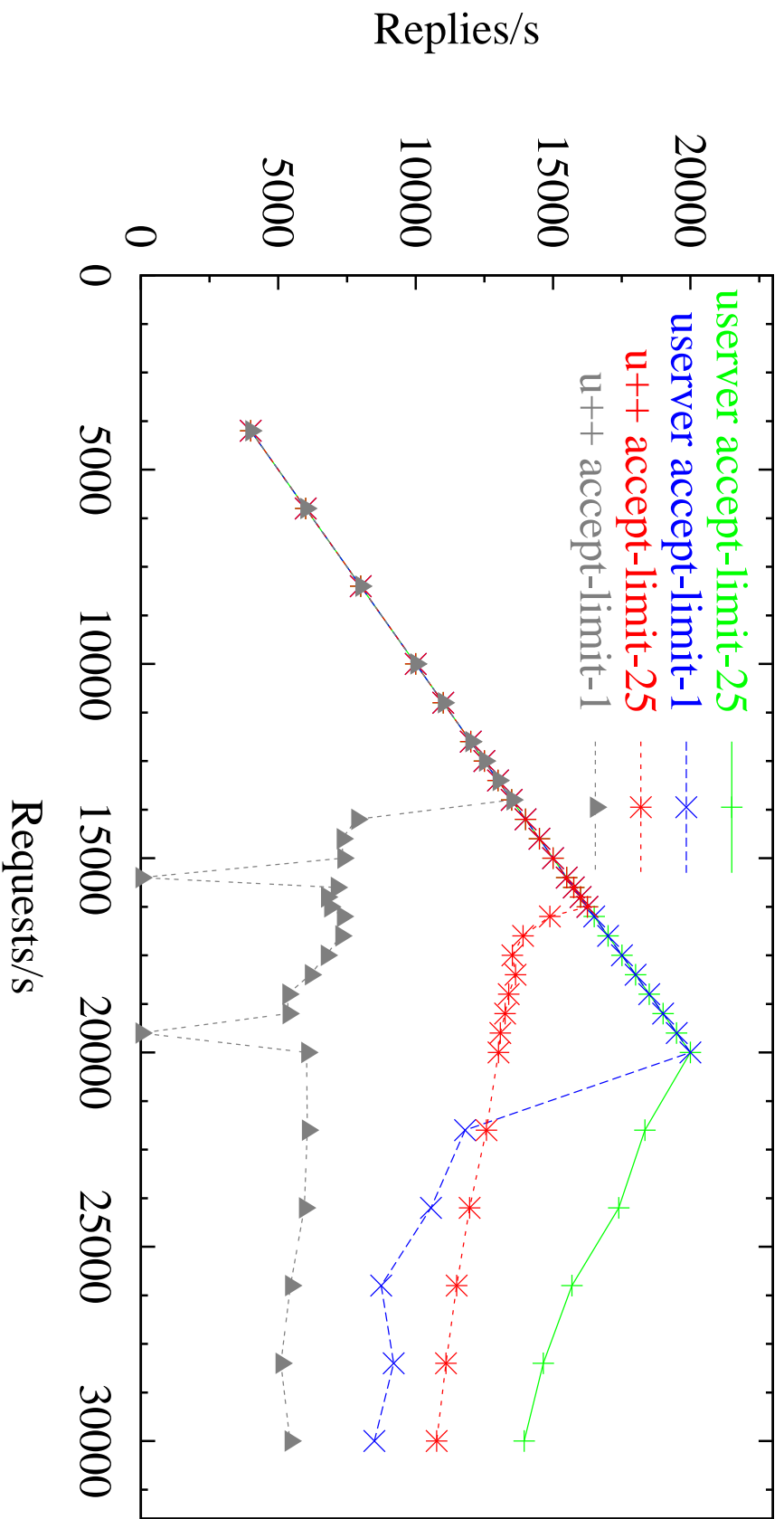


- ports: **Linux IA-64**, Linux IA-32(P3)/AMD, Solaris SPARC
- future work:
 - SCALABILITY, SCALABILITY, SCALABILITY, ...
 - extend statistical metrics
 - * achieve 95% of qprof/VTune non-kernel functionality
 - extend exact metrics
 - port to Intel P4 PMU
- <http://plg.uwaterloo.ca/~usystem/MVD.html>

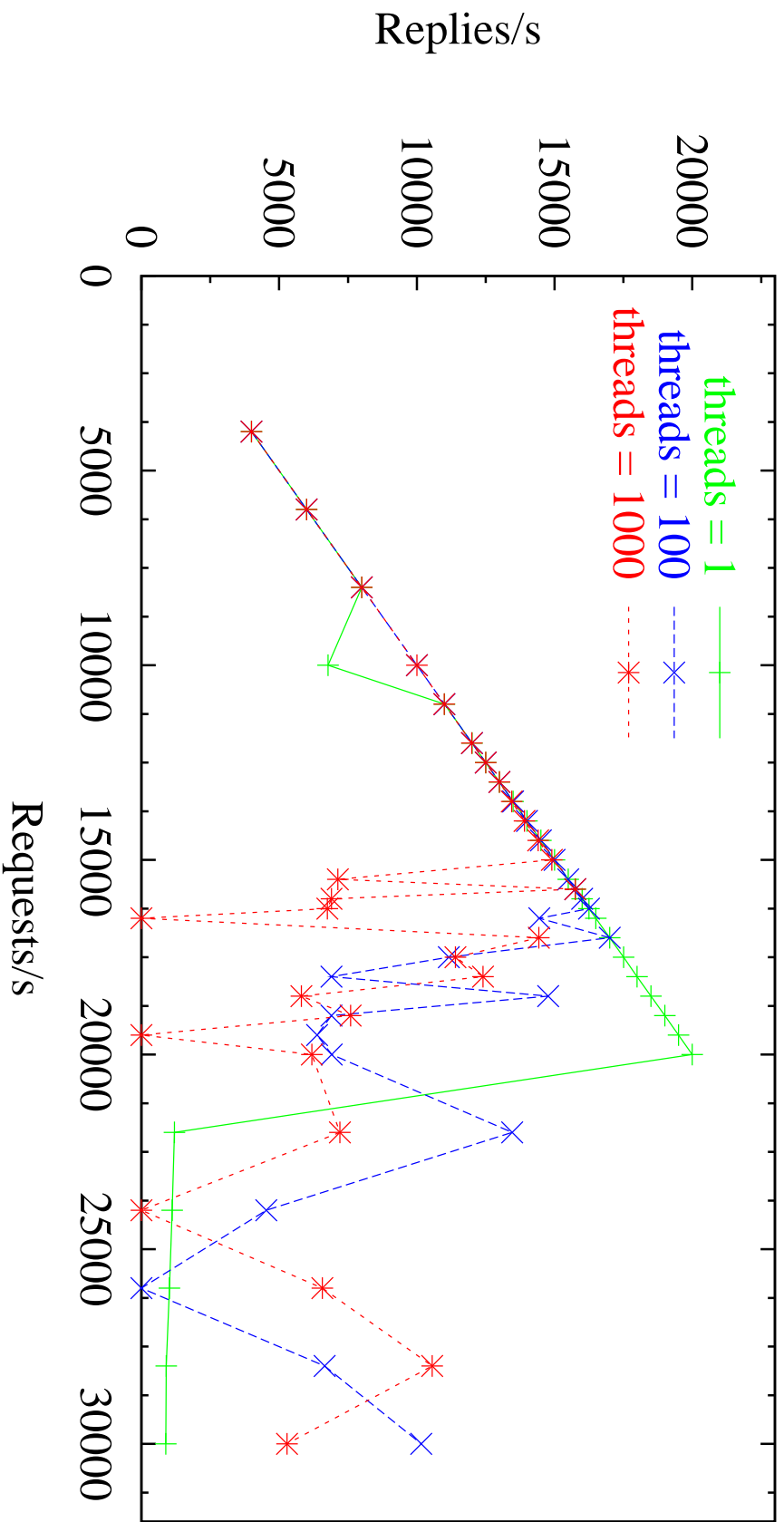
Threaded Web-Servers

- use μ C++ threads and non-blocking I/O to build high-performance web-server
- benefit and compare with event-driven μ server web-server
- threaded web-server: disadvantages/advantages
 - inherently slower than event-driven
 - amenable to additional processing of web data
- two approaches:
 - thread per connection
 - multi-threaded pipeline

Baseline Comparison : μ server / μ C++



Thread / Connection



- preliminary results:
 - threaded web-server within 10% of event-driven server
 - μ C++ non-blocking I/O and object model allows significant simplification in writing server (150 lines)
- future work:
 - multi-threaded pipeline
 - improve behaviour at saturation
 - understand oscillations
 - larger workloads
 - dynamic content
 - scalability testing

Demonstrations Available

- see μ C++ and μ Profiler at University of Waterloo poster

Debugging

