Research in Operating Systems - FlexSC

FlexSC: Flexible System Call Scheduling with Exception-Less System Calls

L. Soares and M. Stumm, U. Toronto

OSDI 2010

Synchronous system call is a (bad) legacy

- System calls are the defacto interface to the OS
- Basic model
  - Write arguments to appropriate registers
  - Issue special instruction that raises a synchronous exception
  - Yielding user-mode execution to kernel-mode exception handler
- Key points
  - Use processor exception to communicate w/ the kernel
  - Enforces synchronous execution model
- Expensive!
  - Direct costs – mode switch
  - Indirect costs – pollution of processor structure

The costs of (synchronous) syscalls

- Mode switch cost (direct cost)
  - Time to execute a syscall instruction in user mode
  - Resume execution in kernel mode and
  - Return control back to user mode
    - Require flushing user-mode pipeline, saving registers, …
  - Mode switch cost - to enter and leave 150 cycles (79+71)
    - Compare with 250 cycles for cache-miss memory access
- System call footprint (indirect cost)
  - Process structure pollution – user-mode state replaced by kernel-mode state
  - Processor structures: L1 data and instruction cache, TLB, branch prediction tables, prefetch buffers, unified caches L2 and L3 …

The costs of (synchronous) syscalls

- System call footprints measured with a high IPC workload from SPEC CPU 2006 benchmark
  - Collected using HPC triggering infrequently
- For processor structures, numbers represent entries evicted

Syscall	Instruc	Cycles	IPC	i-cache	d-cache	L2	L3	d-TLB
Stat	4972	13585	0.37	32	186	660	2559	21
Pwrite	5689	31285	0.18	50	373	985	3160	44
Open+write+ close	9921	32815	0.39	78	481	1462	5105	49

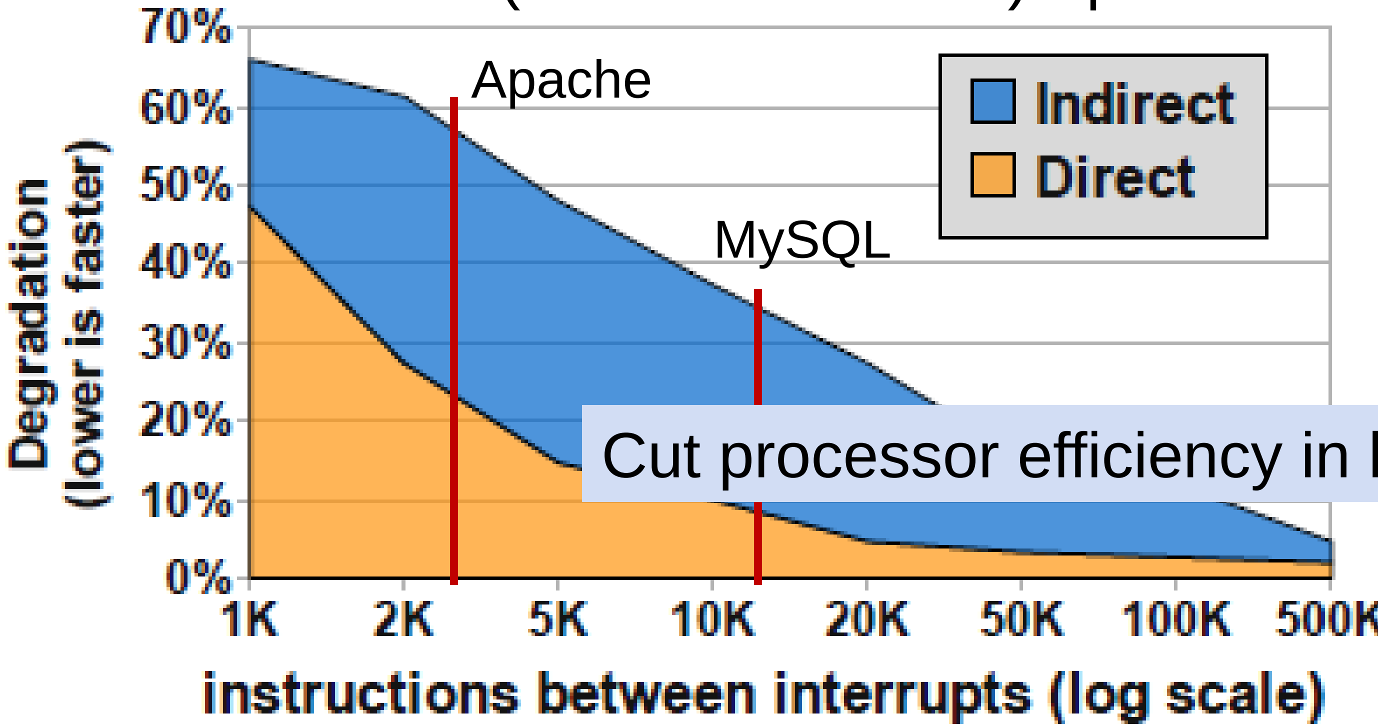
Half to full d-cache!

Much larger than L1 due to L2 & L3 more aggressive preftecting

Synchronous syscall impact on user IPC

- At the end, the measure of cost that matters
  - Direct cost was measure issuing a null system call; indirect is the diff

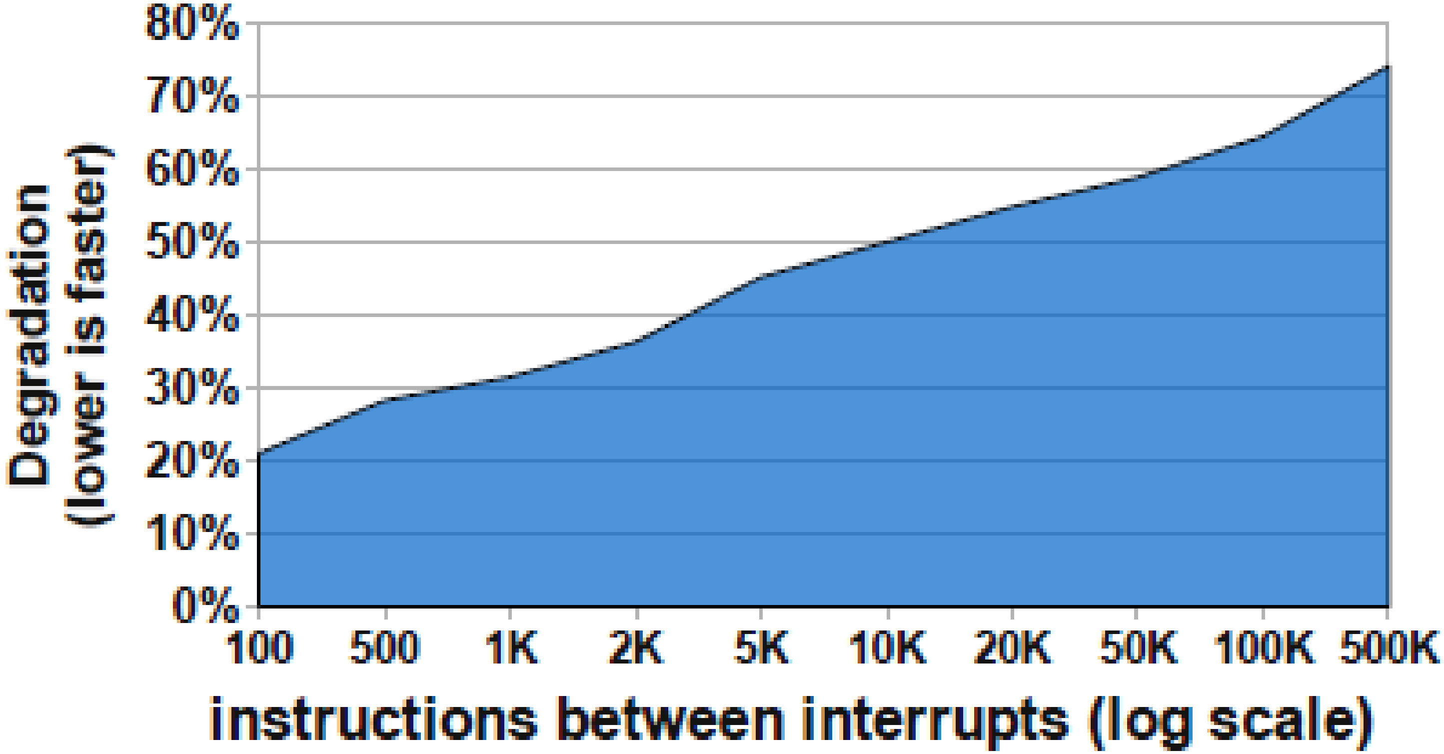
Xalan (SPEC CPU 2006) - pwrite



Cut processor efficiency in half!

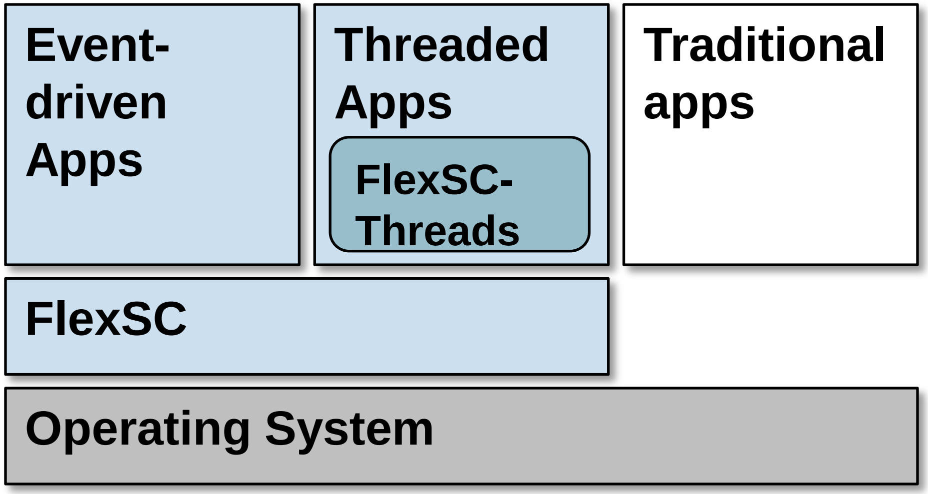
Synchronous syscall impact on kernel IPC

- Lack of locality also impact kernel IPC – trend,of course, is opposite
  - More frequent system calls, more kernel state is maintained



FlexSC contributions

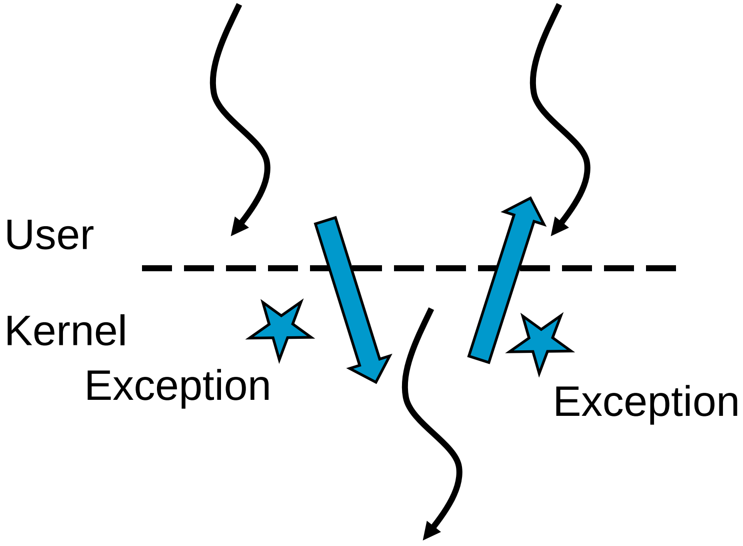
- Quantify impact of synchronous syscalls
- Propose exception-less syscalls and implementation
- Present a thread library to make its use transparent



- Show performance improvement
  - Apache – up to 116%
  - MySQL – up to 40%

Exception-less system calls

- A new OS mechanism – exception-less system calls
- Key idea - remove synchronicity by decoupling invocation from execution



Benefits of exception-less syscalls

- Lower direct costs
  - Fewer mode switches
- Allows for system call batching
  - Reduce indirect costs
- Allows for dynamic core specialization
  - Scheduling a syscall on a core != than where it was invoked
  - Improved spatial locality – lower indirect costs
  - Potentially no mode switches necessary – eliminate direct costs

Exception-less interface

- Interface – a set of memory pages shared bet/ user and kernel mode – syscall pages
- Syscall entries

Syscall number	Number of arguments	status	arg0	…	arg6	Return value

```
write(fd, buf, 4096);
```

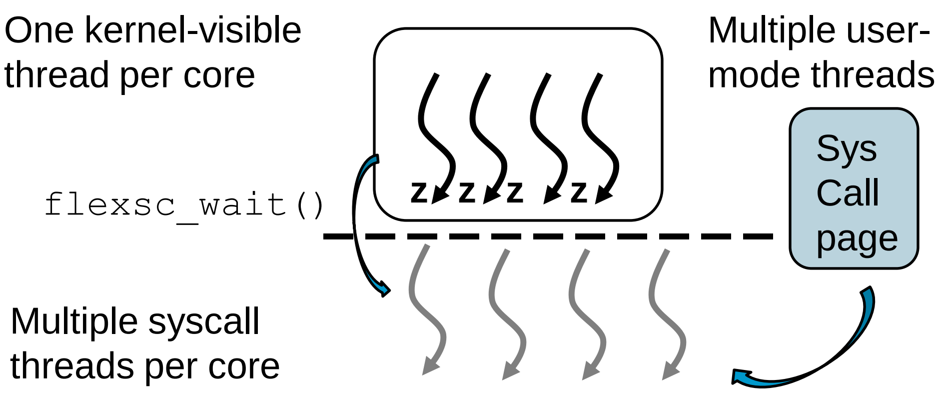
```
entry = free_syscall_entry();
entry->syscall = 1;
entry->num_args = 3;
entry->args[0] = fd;
…
entry->status = SUBMIT;

while (entry->status != DONE)
      /* do something else */;
return entry->return_code;
```

Interface and syscall threads

- Two new system calls
  - `flexsc_register` – register process wanting to use FlexSC
  - `flexsc_wait` – When user-space thread has nothing else to do but wait for at least one return – tell the kernel
- System call executes in the virtual address of the invoking process
  - `flexsc_register` creates syscall threads (cloned from the registering process)
- To maintain syscall blocking model
  - Create multiple syscall threads per process (as many as entries in the syscall page)
  - Only one is active per app/core
  - When thread needs to block, wake up another one

FlexSC thread library

- Get the benefits without (the costs) changing the interface
- M-on-N thread library
  - POSIX compliant, binary compatible with Linux NPTL
  - One kernel visible thread per core, many user threads per kernel
- Redirects system calls (libc)
  - Posts exception-less syscalls to syscall page
  - Switches to another user-level thread
  - If run out of ready user-mode threads
    - Check syscall page for completed entries so that it can get result
    - As a last resort – invoke `flexsc_wait`

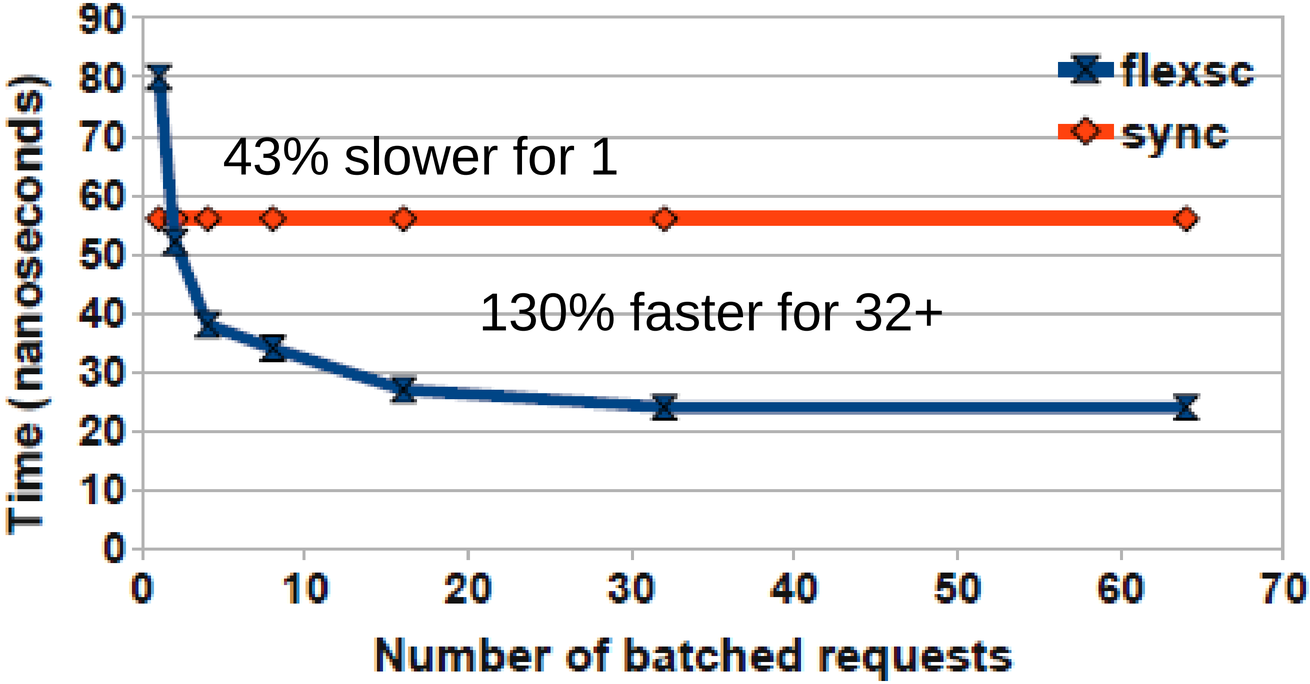


Evaluation

- Linux 2.6.33
- Nehalem (Core i7) server, 2.3GHz
  - 4 cores on a chip
- Clients connected on 1Gbps network
- Workload
  - Sysbench on MySQL (80% user, 20% kernel)
  - ApacheBench on Apache (50% user, 50% kernel)
- Default Linux native POSIX threaded library, NTPL (synch) vs. FlexSC-Threads (flexsc)
- Values reported are avg or 5 runs

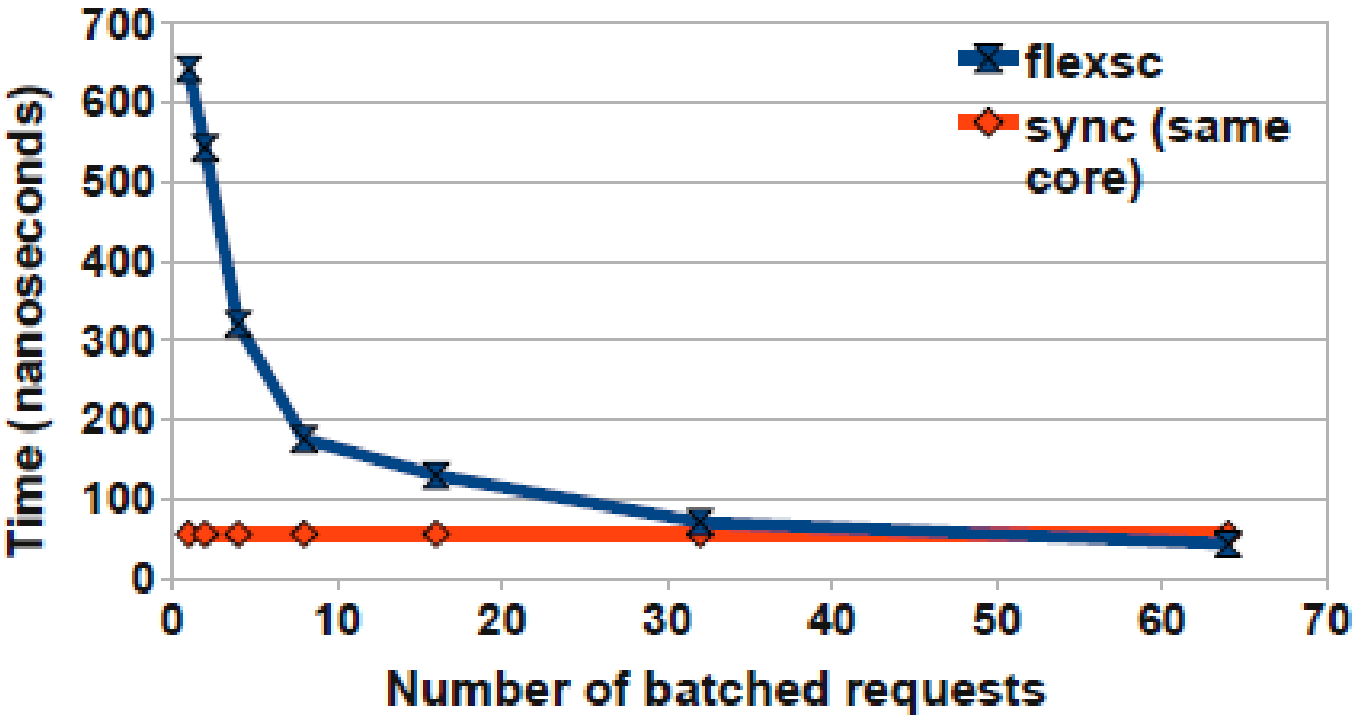
Overhead

- Overhead of execution a exception-less syscall
  - Switching to syscall thread and back to user thread
  - De-marshaling args and retrieve return from syscall page
  - To measure this – microbenchmark using getppid()
    - Small user- and kernel- footprint, what's left is direct cost



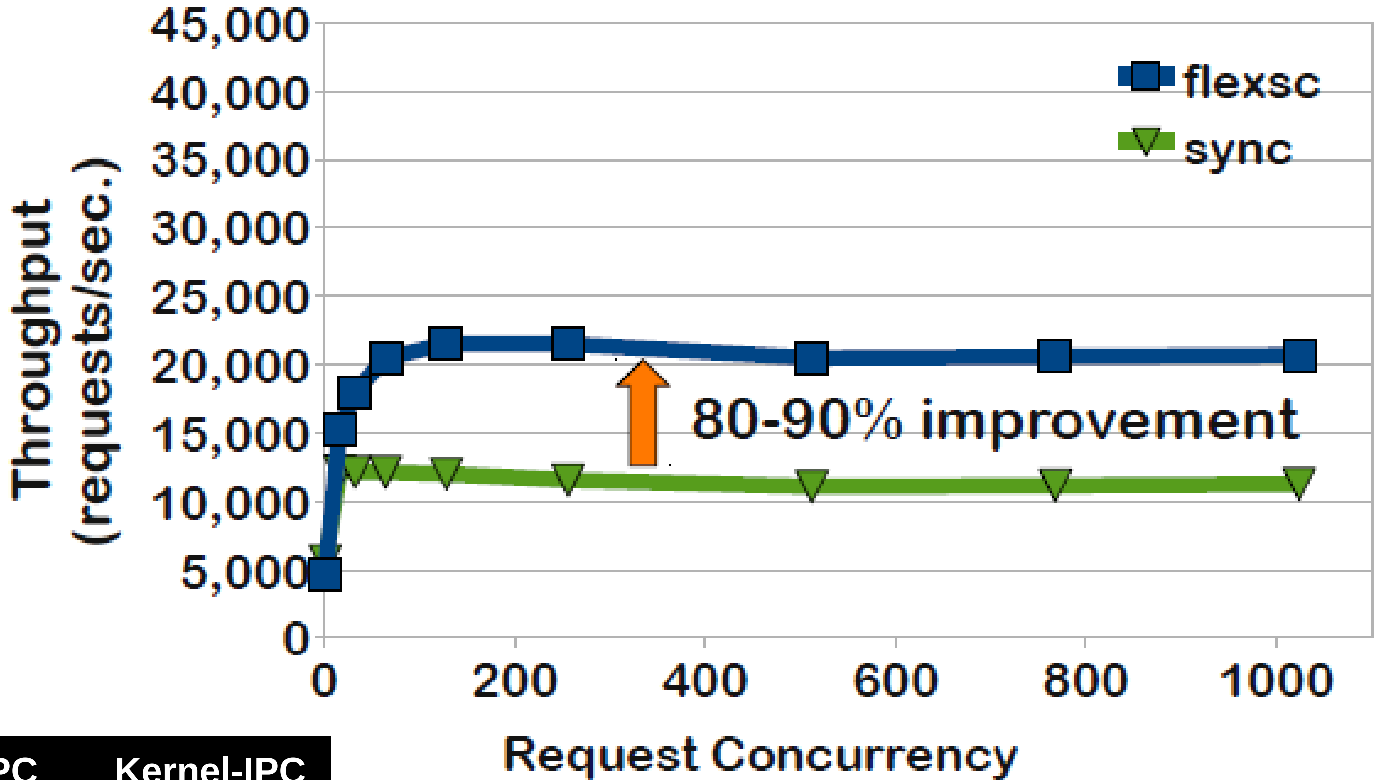
Overhead

- Large overhead to execute on a remote core
- Remote execution requires sending a interprocessor interrupt to wake up remote syscall thread
  - Worst case – not currently executing syscall thread there



ApacheBench throughput (1 core)

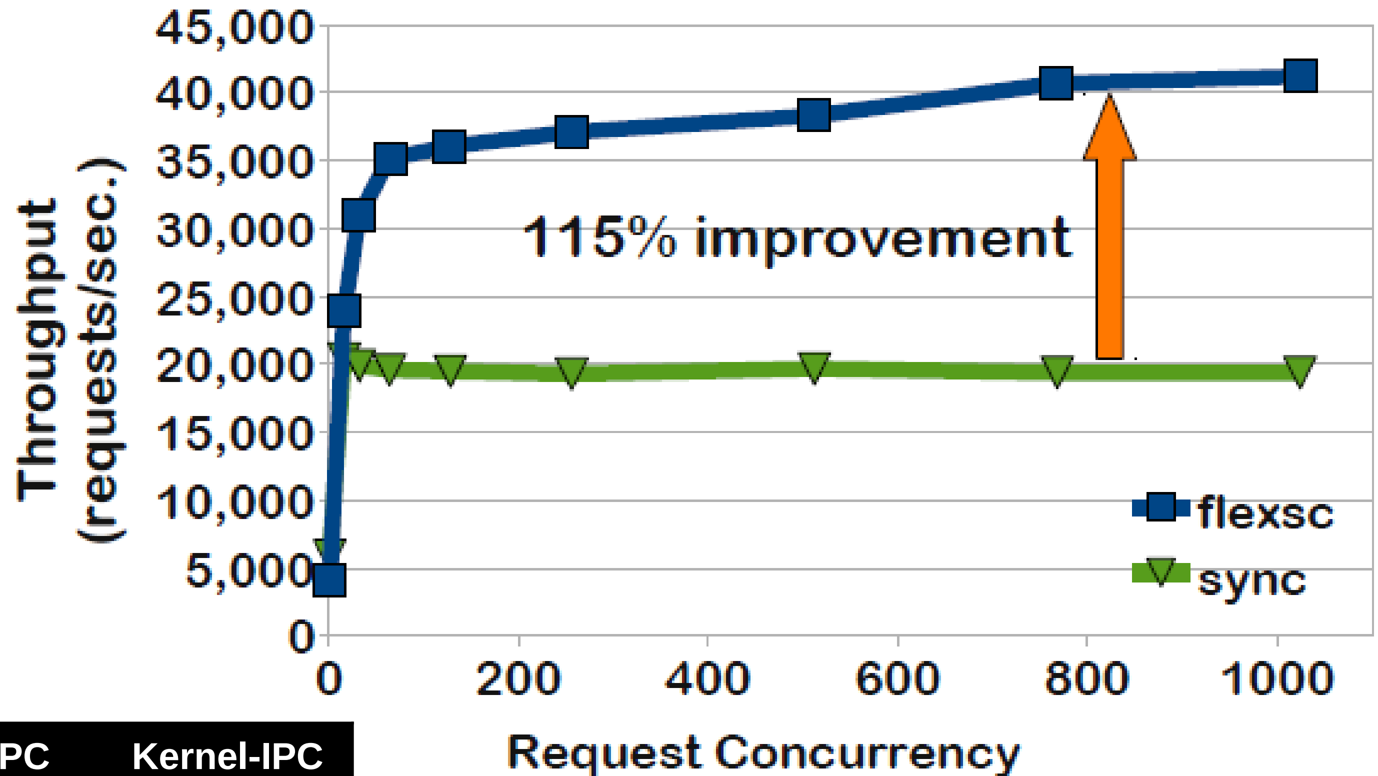
- Apache performance with ApacheBench workload
  - 1 core – FlexSC uses syscall batching



Apache	User-IPC	Kernel-IPC
Sync	0.48	0.45
Flexsc	0.94	0.94

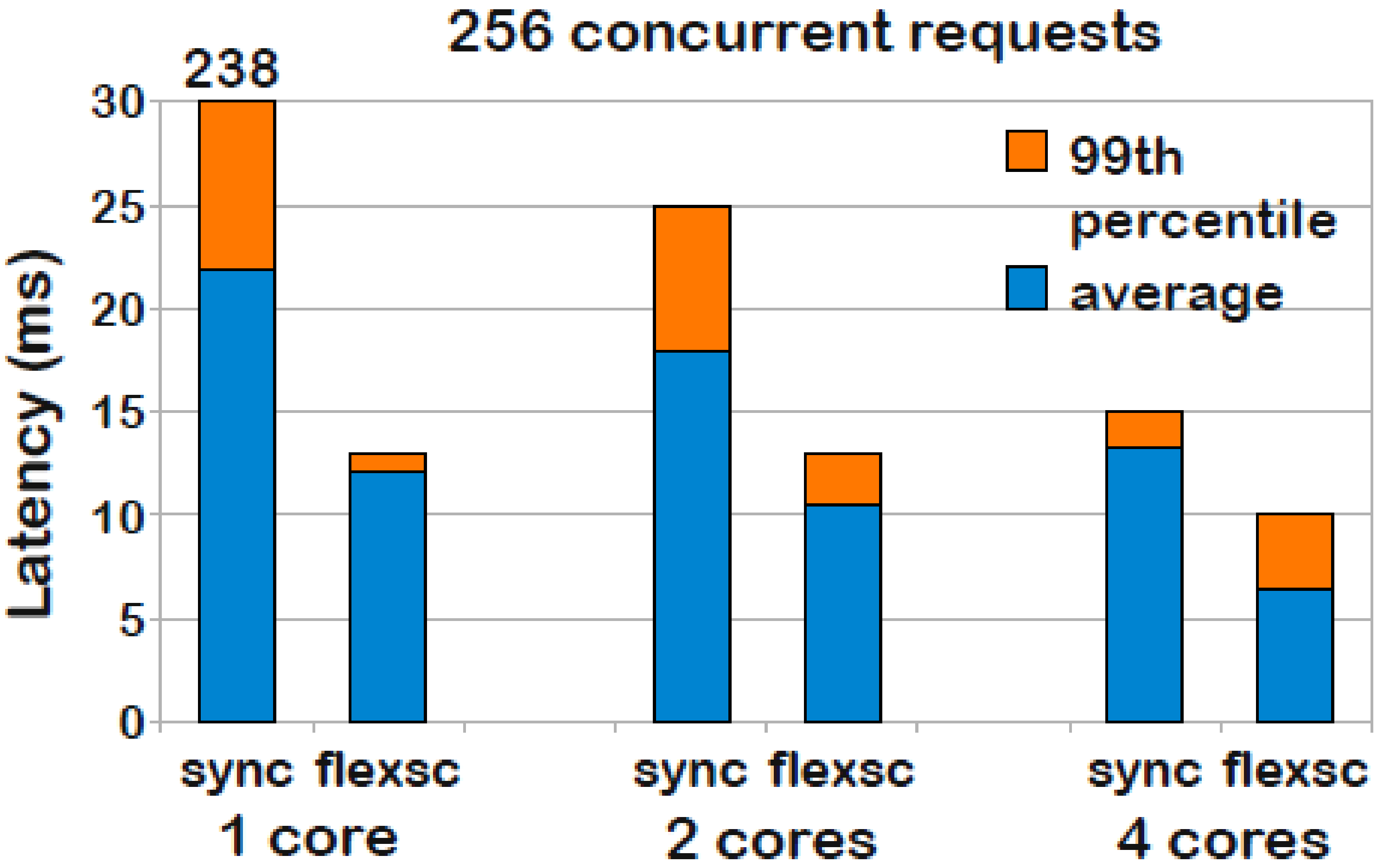
ApacheBench throughput (4 cores)

- On multicore, redirect syscalls to maximize core locality
- Disparity between throughput and IPC improvement
  - Benefit from localized kernel exec – reduced contention for locks



Apache	User-IPC	Kernel-IPC
Sync	0.45	0.43
Flexsc	0.74	0.76

Apache latency per client request



Summary

- Traditional syscall degrade server performance
  – Biggest problem is pollution of processor structures
- Exception-less syscalls - flexible and efficient syscall execution
- FlexSC-threads to use them w/o modifying apps
- Large improvements on throughput and latency of benchmarked apps
- Future work
  – Scheduling of syscalls (time and space)
  – Exception-less syscalls for what they were originally meant – low-latency comm. between user and kernel space with hyper-threaded processors
  – …