Surviving OS Failures in Persistent Memory

David Fiala, Frank Mueller, Kurt Ferreira, Christian Engelmann

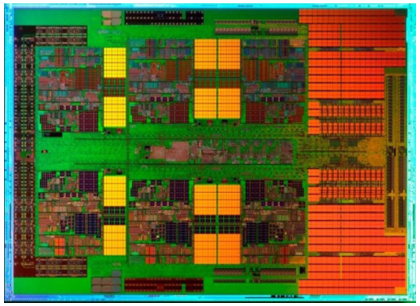
North Carolina State University

Sandia National Laboratories

Oak Ridge National Laboratory

Why Things Can Go Wrong

- Trend in Micro-Architecture:
  - Miniaturization increased chip density (fabs)
  - Increases sensitivity to bit upsets / faults
  - On a PC: ~50 years MTTF → not a problem
    - MMTF: mean time to failure



Istanbul Opteron die
(Source AMD)

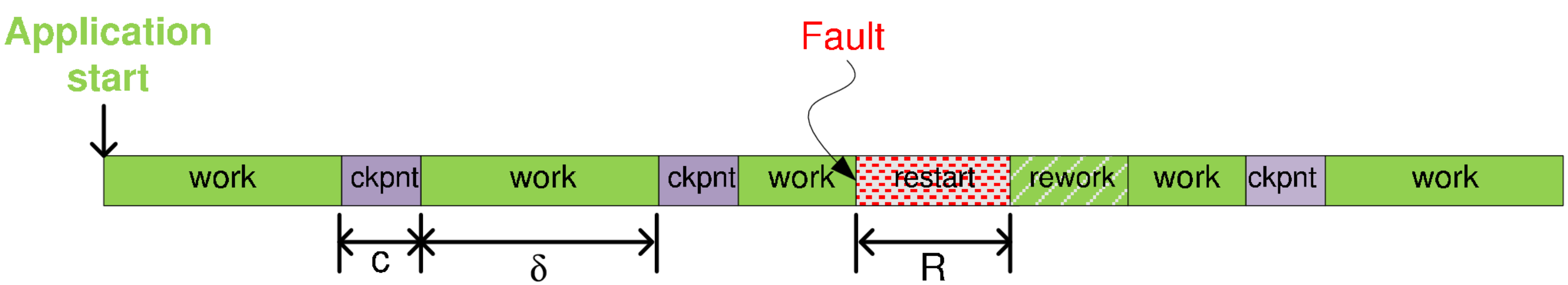
- Data Center / Cloud / High-performance computing:
  - Increasing number of storage / nodes / cores → more faults
  - Power management more critical
    - Lower voltages to reduce power (but also Turbo boost)
    - Higher likelihood of single event upsets (bit flip)

→ **MTTF decreases as cores, power, and density grows**

Case Study: Resilience in HPC

- HPC: 10k-100k nodes
  - Some component failure likely
  - System MTBF becomes shorter
  - Processor/memory/IO failures
- Currently FT exists, but…
  - Not scalable
  - Mostly reactive: process checkpoint/restart
  - Restart entire job → inefficient if only one/few node(s) fail

System	# CPUs	MTBF
ASCI White	8,192	5/40 hrs
Google	1,5000	20 reboots/day
ASC BD/L	212,992	7 hrs
Jaguar	300,000	5/52 hrs



Silent Data Corruption

- Silent Data Corruption (SDC) → bit flips in
  - Storage or CPU cores
  - Some not detectable / correctable
  - Undetected → invalid results, app doesn't stop
  - Severe problem for today's large-scale simulations

- Memory bit flips correctable by ECC
  - Each ECC algorithm may have an upper limit of bit flips
  - Uncorrectable for an instant reboot → **or becomes SDC**

Undetectable errors are expected to occur once or twice per day on ORNL's Jaguar Supercomputer [Geist, Monster in Closet]

SDC Protection

- Hardware: ECC (error correcting/checking codes)
  - SECDED: Single error correct, double error detect
    - 3+ errors undefined!!
  - 8% of DIMMs experience uncorrectable errors [Schroeder]
  - Triple bit error frequency not entirely understood
- Software:
  - Algorithm-based FT (i.e., matrix protection [Huang])
  - Duplicated instructions, registers, memory, etc. [Rebaudeng] [Oh][Reis]
  - Control flow checking [Oh]
  - Background scrubbing [Shirvani]

Generalized Protection is Desirable

- Redundancy: message passing applications only
  - Requires 2x or 3x resources, but effectively 100% coverage

    redundancy becomes baseline

    comparison for 100% detection and/or correction.

- Algorithmic Fault Tolerance → *non-trivial!*
  - **Often difficult to develop**
  - Even so, not comprehensive (i.e., some memory unprotected)

- Our motivation: provide SDC protection to any HPC class of application and operating system

  → **allow developers to focus on efficient algorithms, not resilience**

Application Runtime Dependencies

- Compiled application:
  - Its own code
  - Its own data
  - Libraries (static or shared)
- In HPC: MPI library is unique → handles interface between application and OS's network interface to provide communication with peers
- Operating System (OS)
  - OS abstracts all devices, memory management, etc.
  - **Why protect OS?** → Any failure causes "panic", loss of all unsaved computation. OS remains the last unprotected piece

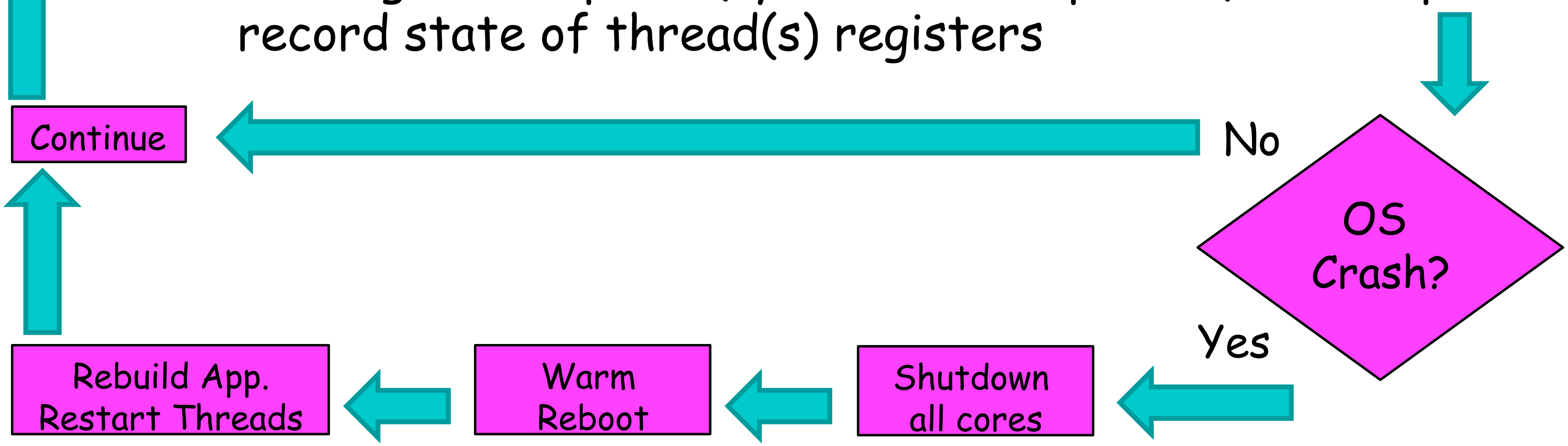
The state-of-the-art OS crash recovery is to simply reboot.

Mini-Ckpts: Contributions

- Objective: Let app survive if OS fails
- Design of Mini-Ckpts:
  - Identify minimal process state @ failure
  - Identify common instrumentation points in OS to save state
  - Warm reboot OS on failure, preserve app and continue exec.
- Implementation:
  - Process protection from kernel failures at syscalls
  - App lives in persistent memory
- Evaluation:
  - cost of mini-ckpts and warm-rebooting a failed OS
  - application survival for injected kernel faults
    - with OpenMP (multithreaded applications)
    - with MPI (message passing applications)

Mini-ckpts Overview

- Requires specialized kernel
- Protection
  - Checkpoint (serialize) structures describing a process
  - Migrate memory to persistent region (survives warm reboot)
  - continue execution…
    - During interruption (syscall, interrupt IRQ, interrupt NMI) record state of thread(s) registers



Supported Features

Feature	Status
Single Threaded Processes	yes
Multi-Threaded Processes	yes
Mutex/Conditions Variables	yes
Process ID	yes
Process UID & GID	yes
Regular Files	yes; but file seek position requires new checkpoint
Unflushed File Buffers	no; Rio File Cache could provide this
Signal Handlers & Masks	yes
Pending Signals	no; any pending are not tracked
Stdin/Out/Err	yes
mmap'd Files	yes
mprotect	yes
FPU State	yes
CPU Registers	yes
Network Connections	no; applications must support restarting connections
Process Credentials	yes
Block Devices	partial; /dev/null, /dev/zero allowed
Special FD's	no; (no signalfd, no eventfd)

Persistent Memory File System

- Anonymous memory stored in page cache ← lost on reboot
- Memory mapped I/O may buffer in kernel
- PRAMFS (Persistent RAM FS)
  - Direct map & execute in place
  - Saved across reboots

Memory Type	Notes
Executable	yes*
BSS Section(s)	yes
Data Section(s)	yes
Heap	yes
Stack	yes (plus each thread's)
Shared Libraries	yes*
Shared Library BSS & Data	yes
vdso & vsyscall	yes, provided by kernel
anonymous mmap'd regions	yes
file-based mmap'd regions	yes*
*Original mapping is migrated to PRAMFS.	

00400000−00401000	r−xp	/hello
00600000−00601000	rw−p	/hello
77beef00000−77beef01000	r−xp	[anonymous][1]
7ffff764a000−7ffff764c000	r−xp	/libdl.so
7ffff764c000−7ffff784c000	−−−p	/libdl.so
7ffff784c000−7ffff784d000	r−−p	/libdl.so
7ffff784d000−7ffff784e000	rw−p	/libdl.so
7ffff784e000−7ffff79d0000	r−xp	/libc.so

→

00400000−00401000	r−xs	/pramfs/temp001
00600000−00601000	rw−s	/pramfs/temp002
77beef00000−77beef01000	r−xs	/pramfs/temp003
7ffff764a000−7ffff764c000	r−xs	/pramfs/temp004
7ffff764c000−7ffff784c000	−−−s	/pramfs/temp005
7ffff784c000−7ffff784d000	r−−s	/pramfs/temp006
7ffff784d000−7ffff784e000	rw−s	/pramfs/temp007
7ffff784e000−7ffff79d0000	r−xs	/pramfs/temp008

- A PRAMFS snapshot at any point in time is similar to a core dump

Processor State Saving During Crash

- 3 Interruption types require instrumentation in kernel
  - Syscall
    - Failure must return *EINTR*; preserve most registers
  - Interrupt (IRQ)
  - Non-maskable Interrupt (NMI)
    - IRQ and NMI both preserve all registers
- During kernel "panic" → Registers previously saved
  - To panic, 1 thread must be in kernel. Any entry point to kernel (above) has already preserved an application.
  - Other threads may be outside kernel → force NMI, save regs
- Panic shutdown protocol: NMI signals → failing core transfers control to core 0
  - Emergency shutdown routines, unpack new kernel, fresh page tables, transfer control to new kernel (like a bootloader)

Application Restoration

- Freshly loaded kernel boots
  - Re-mounts preserved PRAMFS
  - Loads new copy of kernel back in memory
  - Loads services
  - Creates new skeleton process → restore app after crash
    - Memory map cleared out
    - **Old memory from PRAMFS mapped in identically**
    - Same number of threads recreated
    - Kernel schedules threads to run, restores register state
  - Any system call in progress at failure now returns EINTR (Error Interrupted)
    - Restart syscall, and/or rebuild network connections if lost

Case Study HPC: MPI Support

- librlmpi:
  - ReLiable: handles lost messages (network or kernel buffer)
  - libeRaL: tolerates network failures at any point

- Depends on *poll*, *readv*, *writev* syscalls: Detects mini-ckpt restart
  - Reestablishes lost TCP connections
  - Recovery protocol rolls back in-progress lost messages

- Supports: C, Fortran, MPI peer-to-peer, MPI collectives

Experimental Framework

- 4x AMD Opteron 6128
- 1x Intel Xeon E5-2650
- QEMU/KVM Virtual machine environment on AMD and Intel
- Up to 4GB of memory reserved for PRAMFS

- OpenMP: NAS Parallel Benchmarks v3.3 with 8 threads

Benchmark	BT	CG	EP	FT	IS	LU	MG	SP	UA
Class	A	B	B	B	C	A	B	A	A

- MPI: 4 Processes (interconnected with Gigabit Ethernet between AMD nodes)

Benchmark	CG	EP	IS	LU
Class	C	C	C	B

Panic Injection

- Kernel module support to trigger faults
  - Provides *ioctl* syscall taking an argument specifying injection
  - Dereference null pointer
  - Overwrite task_struct members:
    - *fs* , signal handlers, *parent*, *files*
  - Directly call panic

1. Automatic:
   - Shared library providing API to call ioctl
   - MPI: passes rank and iteration number. Environment variables predetermine failure points for specific ranks & times.
2. Manual:
   - Trigger application calls ioctl from command line

Warm Reboot

- Time from kernel panic until
  - (a) kernel is loaded, and
  - (b) software stack initialized from PRAMFS
    - Single largest kernel boot cost: network initialization
- Warm Reboot Total → time at which app may be restored/resumes
- Virtual machines (VMs) do not require initializing physical h/w
  - i.e., network cards

(measured in seconds)	BIOS Boot Time	**Kernel Boot Total**	Network Driver & NFS-Root Mounting	Kernel Misc	Software Stack Total	Cold Total w/ BIOS	**Warm Reboot Total**
AMD Bare Metal	37.4	**5.3**	1.5	4.8	0.7	50.3	**6.0**
Intel Bare Metal	50.8	**6.7**	3.0	3.7	0.7	73.0	**7.4**
AMD VM	—	**0.8**	< 0.2	< 0.6	3.0	—	**3.8**
Intel VM	—	**0.7**	< 0.2	< 0.5	1.3	—	**1.9**

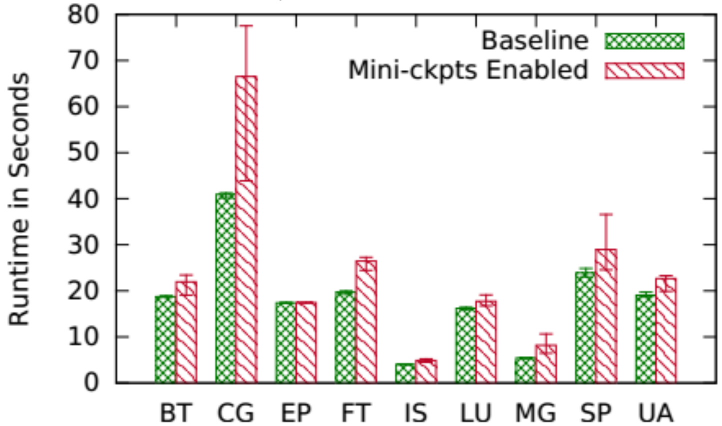
Recovery Testing

- CPU register stress test
  - Modify all registers in deterministic pattern; verify pattern
  - Repeat 100x injections
- FPU stress test
  - Perform floating point add/subtract/multiply/divide
  - Ensure results stay within 10^{-5} of expected value.
  - Repeat 100x injections
- Simple terminal applications
  - vi*, python, sh shell   *terminal must be reset manually

- Regardless of injection type, if it resulted in a kernel panic, then all applications continued execution successfully.

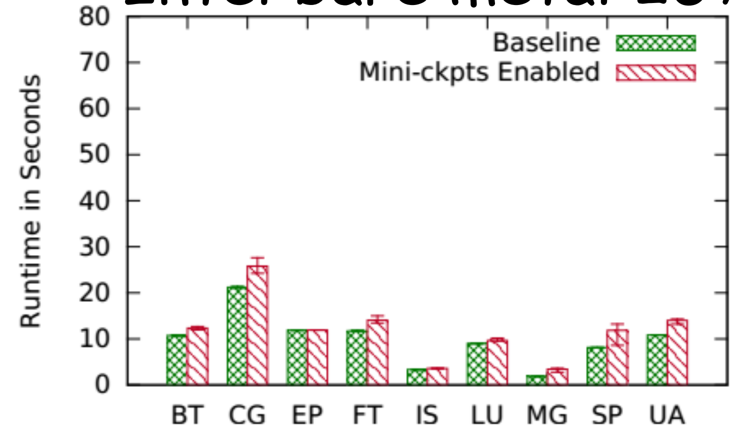
OpenMP Experiments (No Pinning)

AMD bare metal 26%



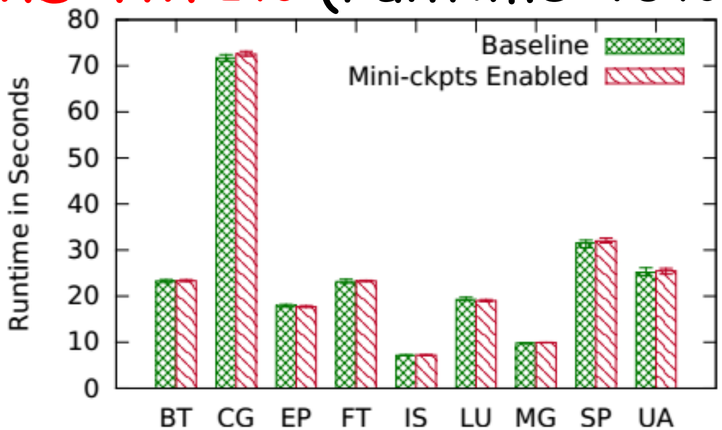
(a) Bare Metal AMD Opteron 6128

Intel bare metal 25%



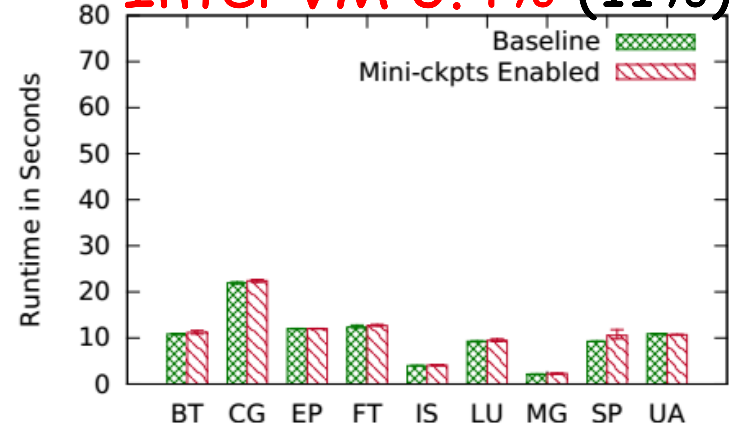
(b) Bare Metal Intel E5-2650

AMD VM 1% (runtime 40%)



(c) KVM Hypervisor on AMD Opteron 6128 Host

Intel VM 3.4% (11%)



(d) KVM Hypervisor on Intel E5-2650 Host

Source of Mini-ckpts Overheads

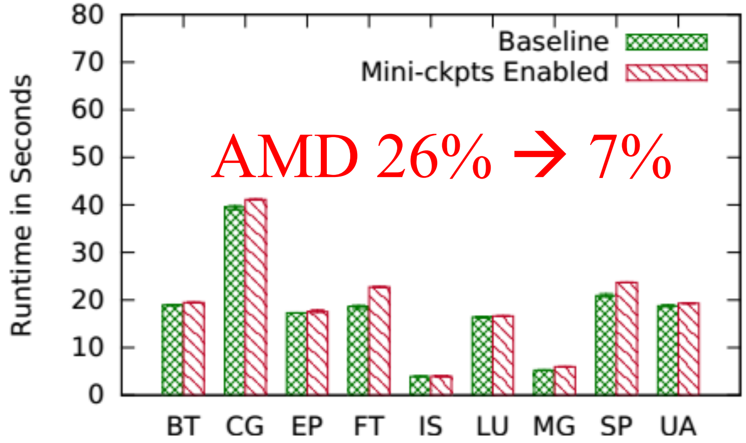
- Mini-ckpts affects applications in two ways:
  - PRAMFS Mappings
  - Instrumented System Calls (investigated second)
- PRAMFS maintains constant physical memory location
  - NUMA architectures experience different latencies by memory controller
- Experiment 1) Microbenchmark: Write 6GB of data to 64MB PRAMFS mmap

Cores	0-3	4-7	8-11	12-16
AMD	1.42	2.04	3.25	3.30
AMD VM	3.2 - 3.4			
Intel	0.90		1.12	
Intel VM	0.95			

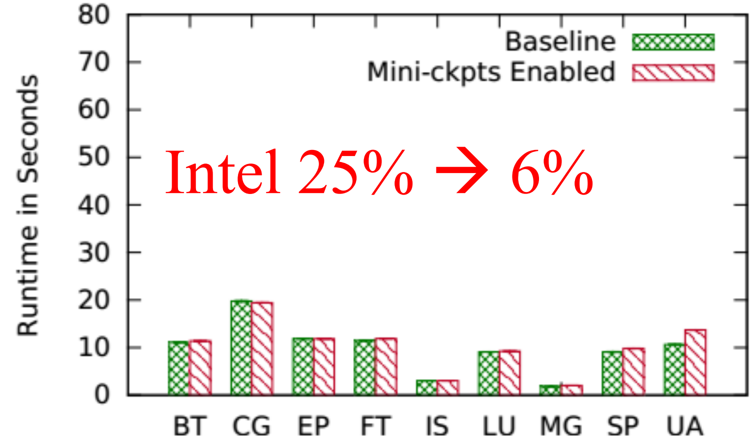
All Times in Seconds

- Experiment 2) Run benchmarks with PRAMFS mappings only (no mini-ckpts enabled)

OpenMP Experiments (Optimal Pinning)

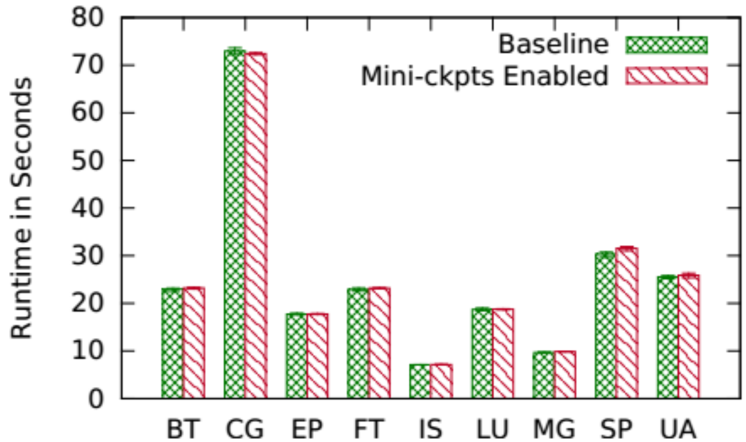


(a) Bare Metal AMD Opteron 6128

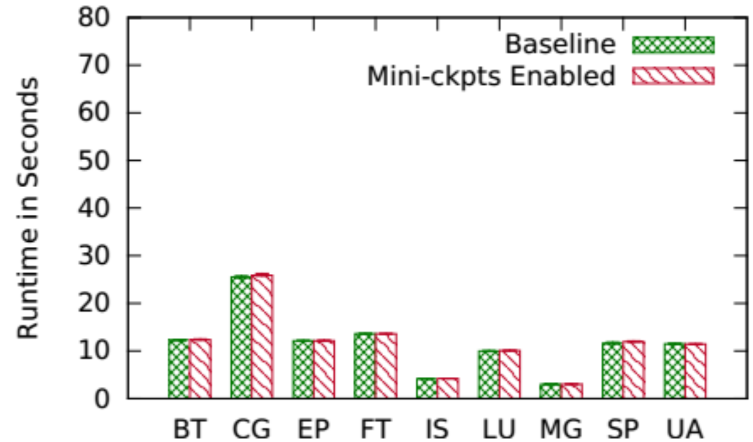


(b) Bare Metal Intel E5-2650

VM hypervisor rescheduled cores regardless of pinning



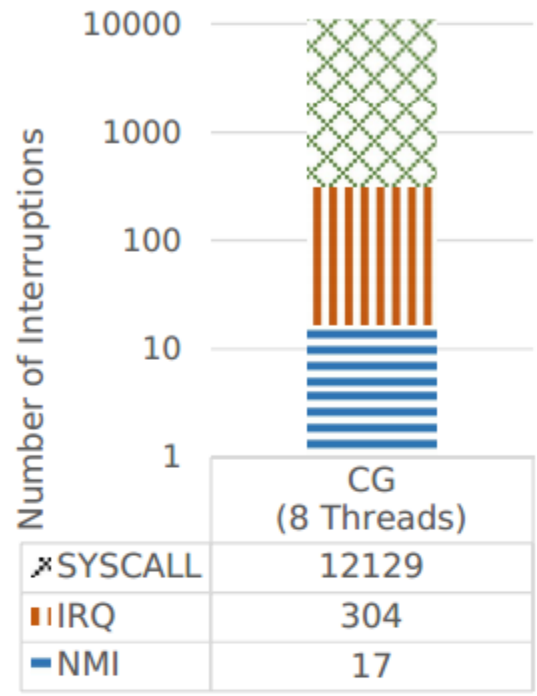
(c) KVM Hypervisor on AMD Opteron 6128 Host



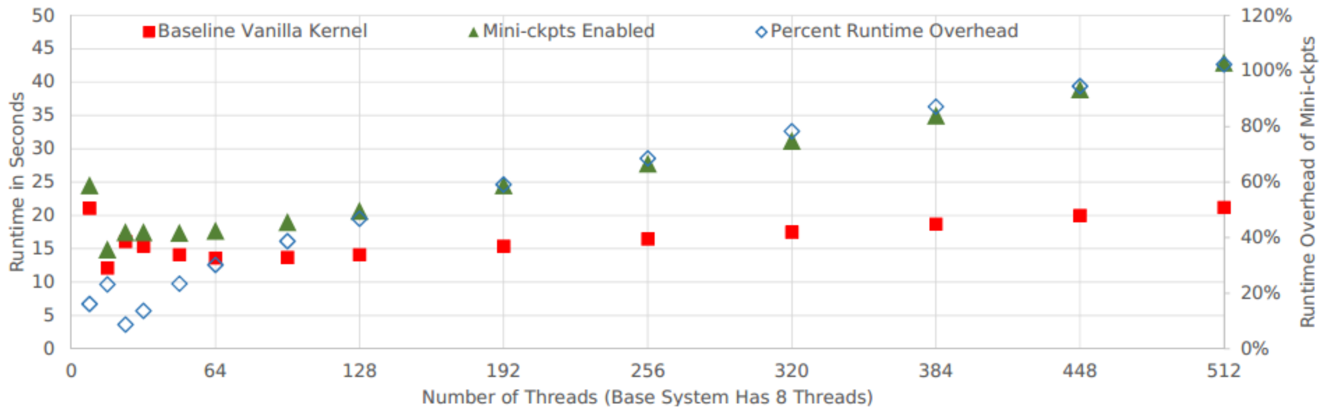
(d) KVM Hypervisor on Intel E5-2650 Host

Extreme Thread Scaling with Syscalls

- Scale NPB CG from 8 threads to 512 threads
  - 32x overcommit of threads to physical cores
  - Predominately calls *futex* syscall during execution
  - Inject panics at highest thread count
    - Recovered successfully
- How does mini-ckpts performance scale?



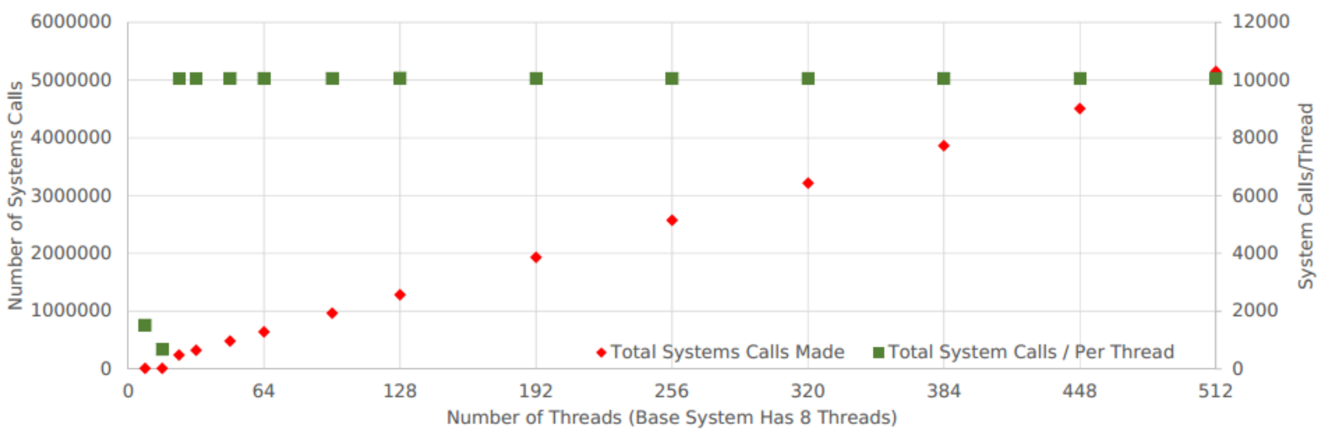
Extreme Scaling → Linear Slowdown



Red left axis: Baseline Runtime

Green left axis: Mini-ckpts Runtime

Right axis: Percent Overhead



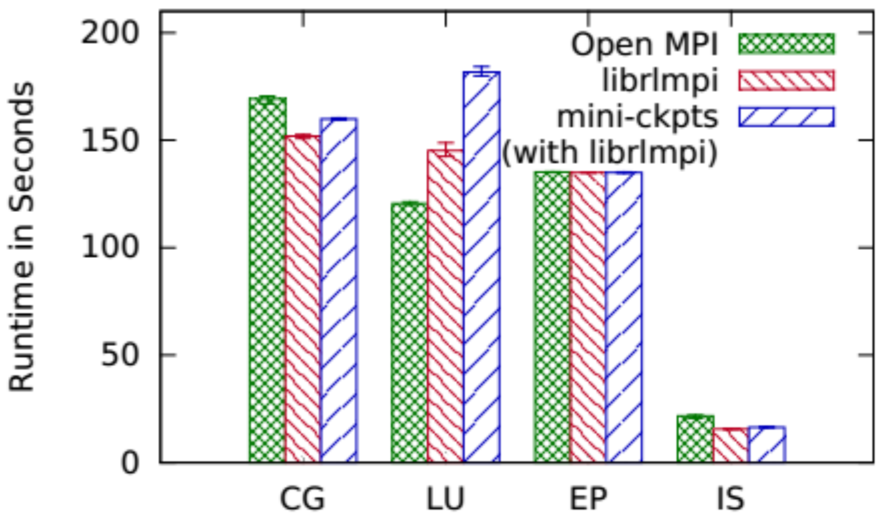
Red left axis: Syscalls per thread

Green right axis: Cumulative syscalls

- Mini-ckpts scales linearly wrt. # syscalls & threads
- Supports 512+ threads

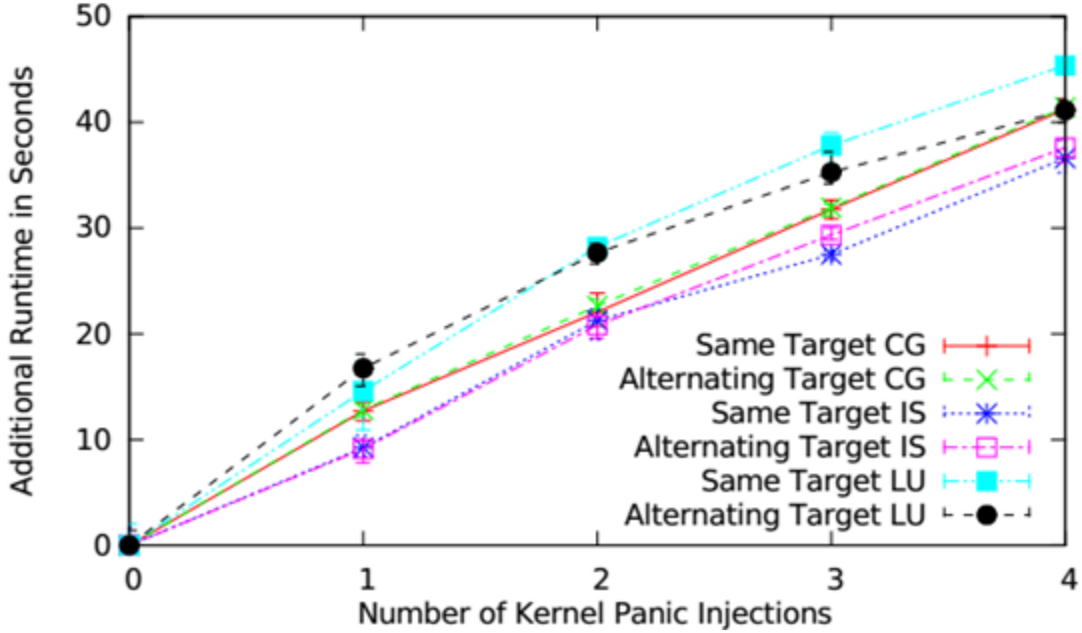
Case Study HPC: MPI Performance

- Failure free evaluation of
  - Open MPI vs. librlmpi
    - Only to demonstrate our prototype is comparable
  - librlmpi vs. librlmpi+mini-ckpts enabled
- NPB MPI Benchmarks
  - CG and IS: (vs. librlmpi standalone)
    - 5% overhead
  - LU:
    - **MPI_Allreduce**
    - 25% overhead
  - EP:
    - 0% overhead



Case Study HPC: MPI Failure Injection

- Injections target same node (1,1,1,1), or alternating (1,2,3,4)
  - X-axis: number of injections, y-axis: additional runtime



Number of Kernel Panics	1	2	3	4
Avg. Time Increase	12.53	11.89	10.76	10.14

All Times in Seconds

- Linear slowdown relative to injection rate
  - **Injection target does not affect outcome**

Additional Kernel Failure Cases

- Memory Allocation Failure
  - Exhausted kernel memory
    - Mini-ckpts **ensures** emergency shutdown does no allocations
  - Hard hangs
    - NMI Watchdog → Hangs while interrupts are off
  - Soft hangs
    - Watchdog timers are being reset, but no progress is made
    - Depends on sanity checking (mini-ckpts **cannot** protect from these unless the kernel subsystem detects a problem)

Related Work

- MVS
  - OS requires recovery routines for services (50% success)
- NOOKS
  - Wrappers around drivers – isolation for the core of OS
- Rio File Cache
  - Write-back file cache in memory (survives a warm reboot)
- Otherworld
  - Specialized crash kernel → warm reboot and "parse" old kernel data structures to recover applications.
    - Corruption in kernel data yields technique ineffective

Conclusion

- Today's OS's not designed with fault tolerance in mind
  - Mini-ckpts provides resilience to appliations if kernel fails
  - Rejuvenates kernel, apps survives in persistent memory (PRAMFS)
- Ckpt/restart is expensive for HPC apps
  - mitigating an OS crash allows **forward progress** w/o restart
- Mini-ckpts identifies key OS changes & structures req'd for resilience
- Warm reboots complete in ~6 seconds, overheads between 5%-8%
  - Both threaded and MPI applications recoverable
  - Scalable in # threads

1st ever transp. OS fault tolerance w/o loss of state

Apps could outlive OS → even if OS instable

Acknowledgement

Supp. in part by DOE/NFS grants, Humboldt fellowship

DOE DE-FG02-05ER25664, DE-FG02-08ER25837, DE-AC05-00OR22725, NFS 0237570, 0410203, 0429653, 1058779, 0958311, 0937908

DOE DE-AC04-94AL85000 (SNL) , DOE DE-AC05-00OR22725 (ORNL) , LBL-6871849 (LBL)

sponsored by the U.S. Department of Energy's Office of Advanced Scientific Computing Research

- NCSU: David Fiala, Frank Mueller
- ORNL: Christian Engelmann
- SNL: Kurt Ferreira

NC STATE UNIVERSITY
Department of Computer Science

Sandia National Laboratories

OAK RIDGE National Laboratory