Towards Real-Time Checkpoint/Restore for Migration in L4 Microkernel based Operating Systems

Sebastian Eckl, M.Sc.

David Werner, M.Sc.

Alexander Weidinger, B.Sc.

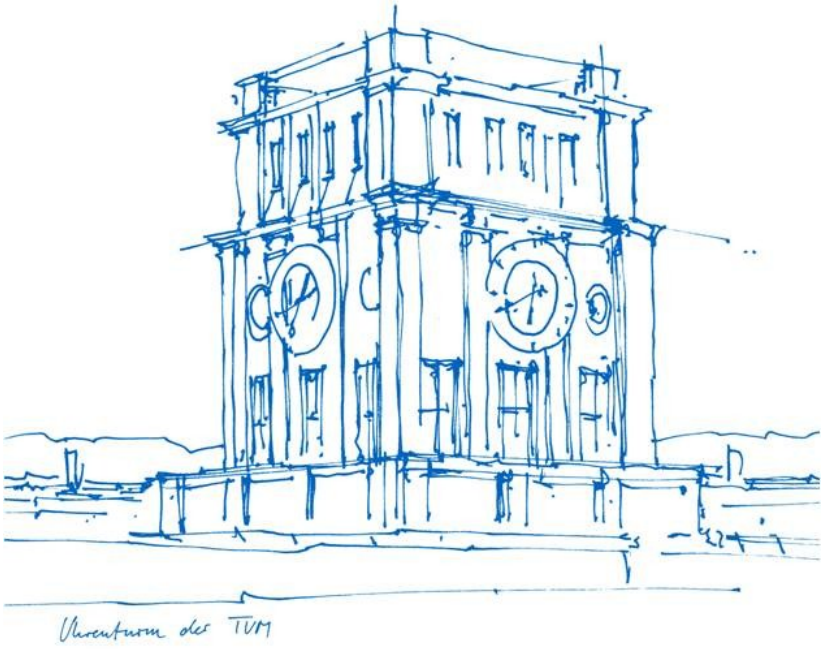
Prof. Dr. Uwe Baumgarten

Technische Universität München

Informatics Department

Chair of Operating Systems (F13)

Renningen, July 9th 2019



Outline

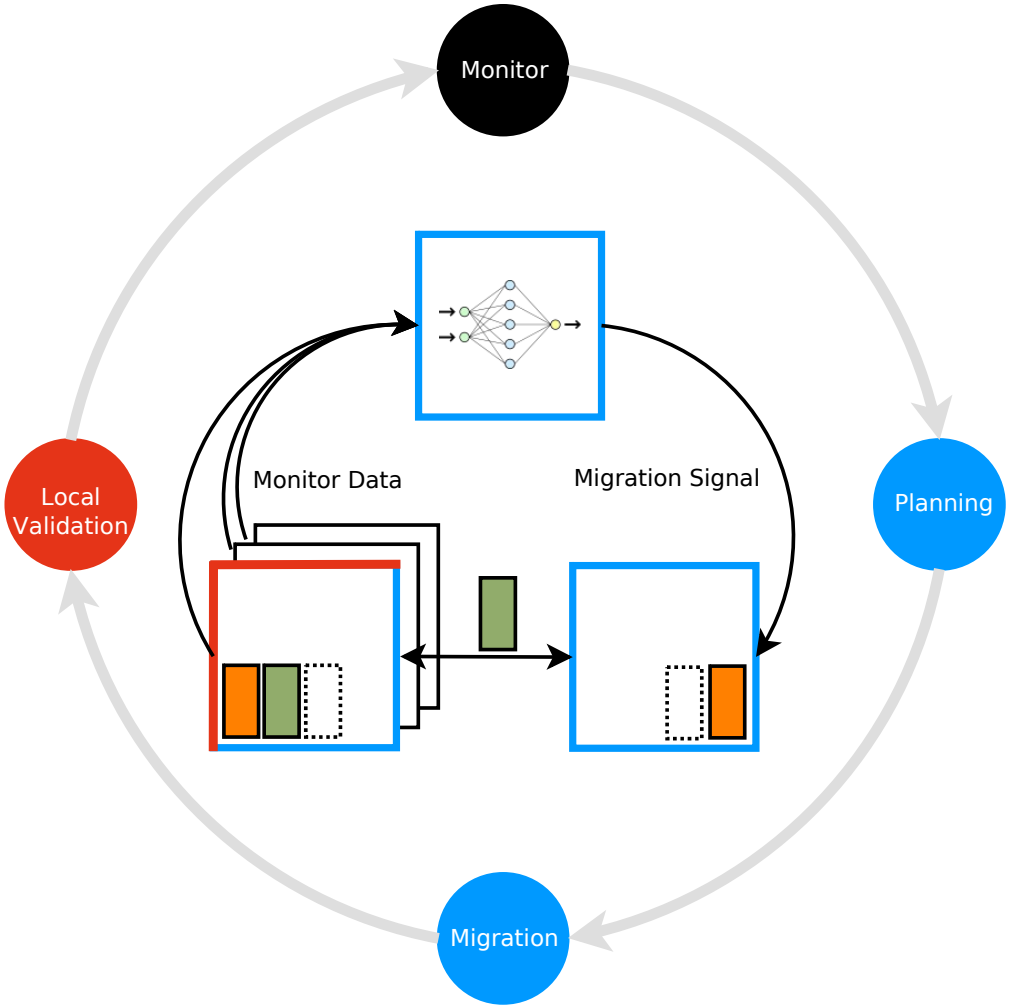
1. Background: KIA4SM
2. Genode OS Framework: Foundation
3. Real-Time Checkpoint/Restore (RTCR)
4. Hardware-based Optimizations
5. Evaluation
6. Ongoing Work
7. Conclusion

1. Vision: Online Reconfiguration via Migration

- **Idea:** Adapt concept of virtual machine (live) migration to the area of CPS

- **Use Case**
  - Fault Tolerance/Availability
  - Efficient Resource Management (Load Balancing/Energy Saving)
  - Scalability

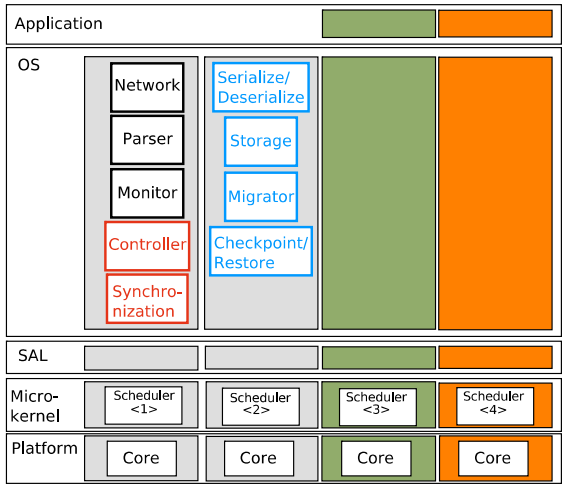
- **Challenge:** Flexibility vs. Reliability/Safety

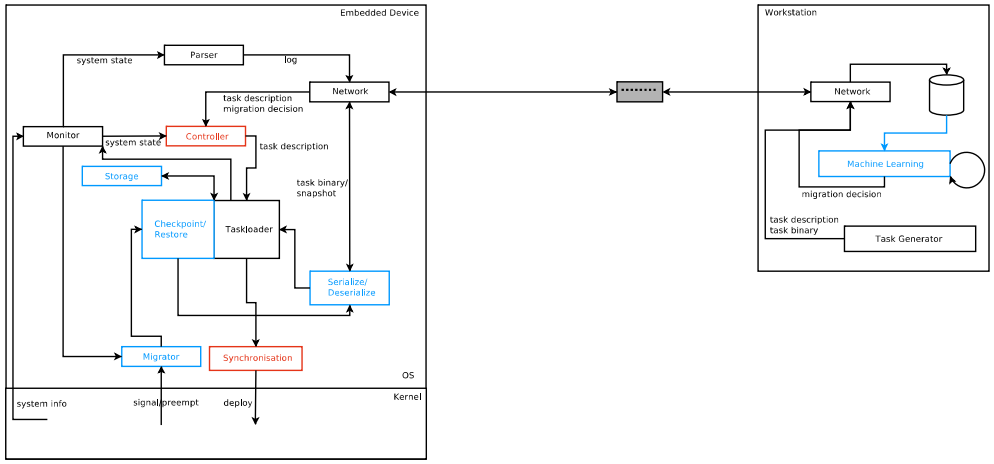


1. Concept & Implementation

- **Operating System (Migration Execution)**
  - Homogeneous RTE (L4 Microkernel)
  - Mixed-Critical Partitioning on Multi-Core
  - Migration Support (Components)

- **ARMv7 & L4 Fiasco.OC / Genode**





2. Genode OS Framework

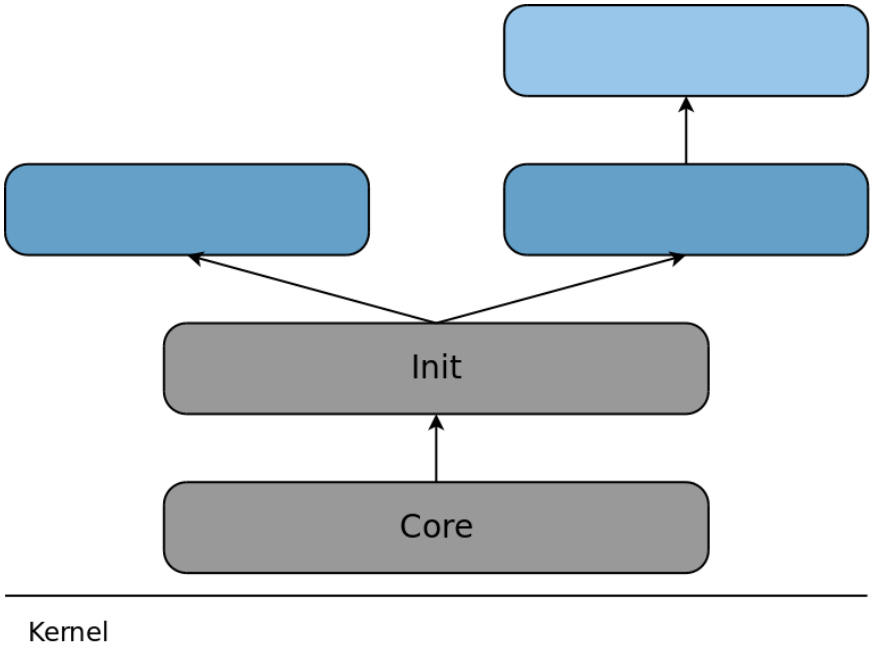
- Component Hierarchy (Minimal TCB)
- Parent/Child-Relationship
- Capability-based Security
- Genode Core Services

Information in this chapter is based on ‚Genode Foundations‘ manual [1]

2. Genode: Component Hierarchy

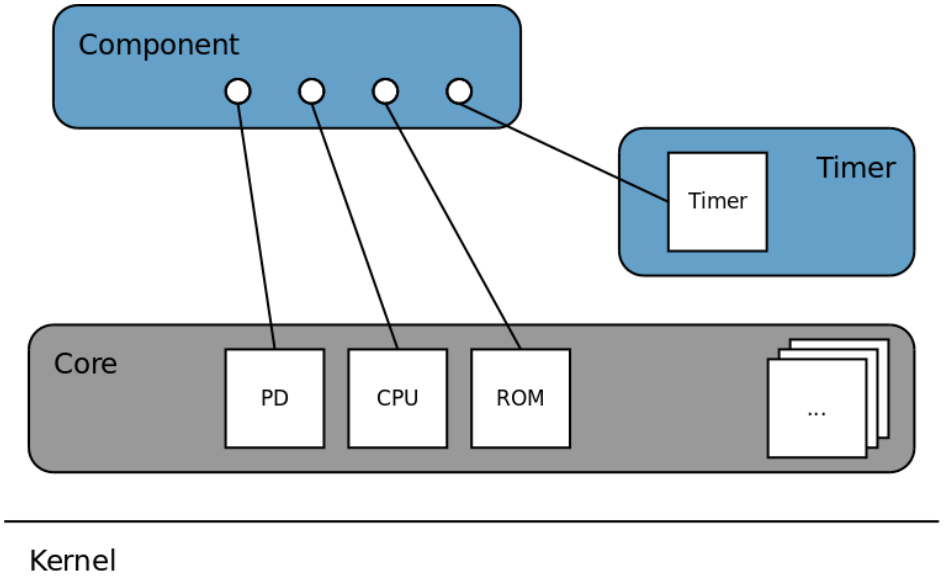
Hierarchy leads to Parent-Child relationships:

- **Responsibility for:**
  - Resource budgets
  - Execution environment
- **Control over:**
  - Lifecycle
  - Relationships



2. Genode: Core Services & Capabilities

- **PD**
  - Represents the protection domain of a component (task)
  - Provides: virtual address space (region map), capability space, resource budgets (memory & capabilities)
- **CPU**
  - Allows creation, manipulation & destruction of threads
  - Assignment of threads to CPU cores (affinity)
- **ROM**
  - Offers access to boot modules
  - Binary is encapsuled in a dataspace



3. Real-Time Checkpoint/Restore (RTCR)

Topics:

- Properties
- Shared Memory Checkpointing
- Redundant Memory Checkpointing
- Incremental Memory Checkpointing

Notes:

- Proof of concept implementations
- Current focus: fast checkpointing procedures
- Current priority: best possible performance
- Future work: Predictability analysis

3. RTCR: Properties

- **Stop/Start mechanism**
  - Consistent snapshot
  - Goal: Minimal stop time
- **Transparency**
  - Components are unaware of checkpoint logic
  - No self-checkpointing of components
- **Security**
  - Conformity to L4 Microkernel design principles
  - Keep minimal TCB
- **Userspace:** Keep C/R logic out of kernel

3. RTCR: Variants and Optimizations

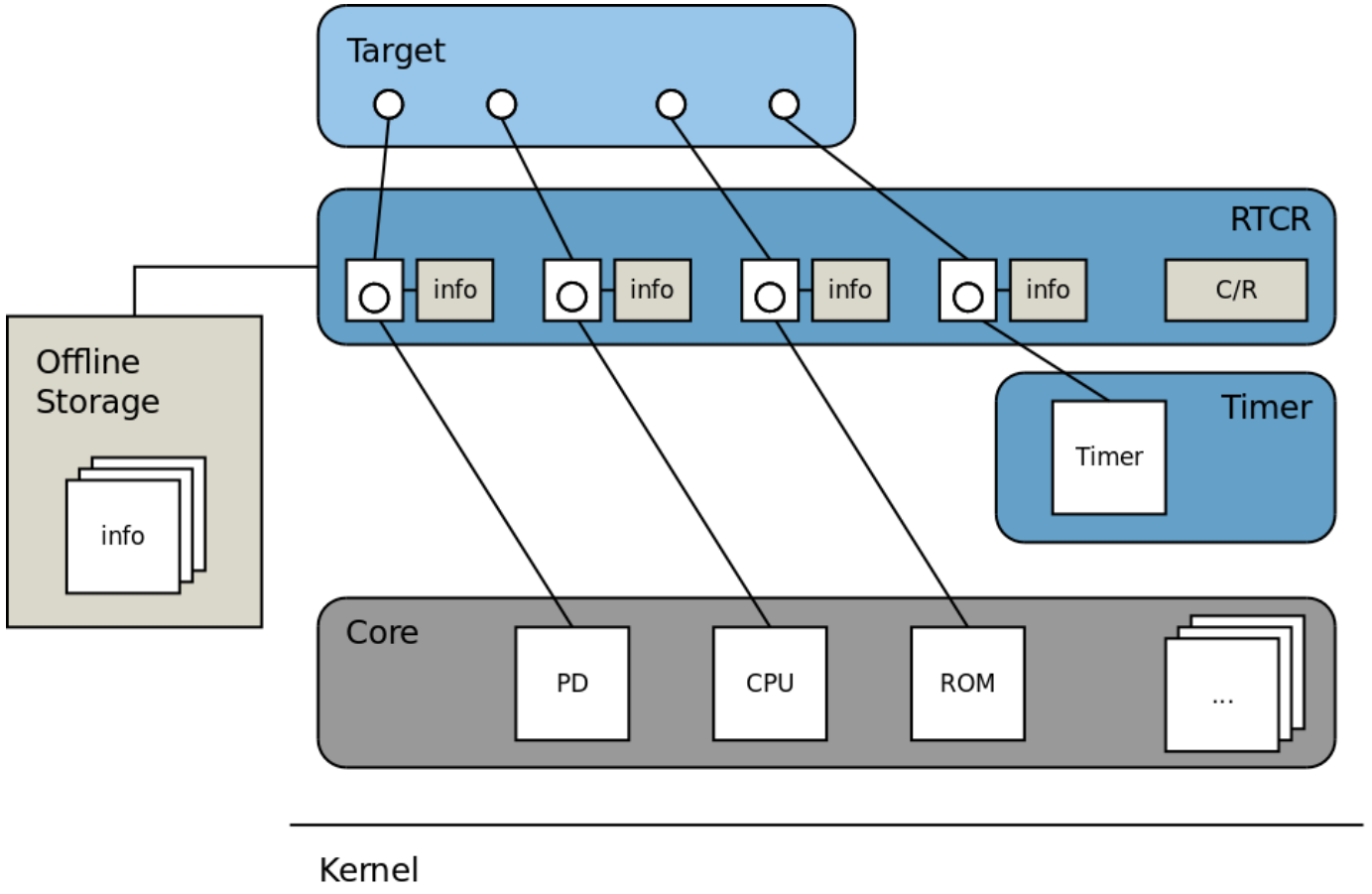
RTCR offers different variants:

- **Shared Memory Checkpointing (SM)** :
  - Explicit copying of target‘s memory contents
  - Target‘s memory is shared with RTCR (parent)
- **Redundant Memory Checkpointing (RM)** :
  - Write emulation to multiple memory areas
  - Creation of redundant copy during runtime of the target
- **Only difference: memory checkpointing mechanism**

RTCR contains software-based optimizations:

- Incremental Checkpointing (Inc)
- Copy-on-write (CoW)

3. RTCR: Overview



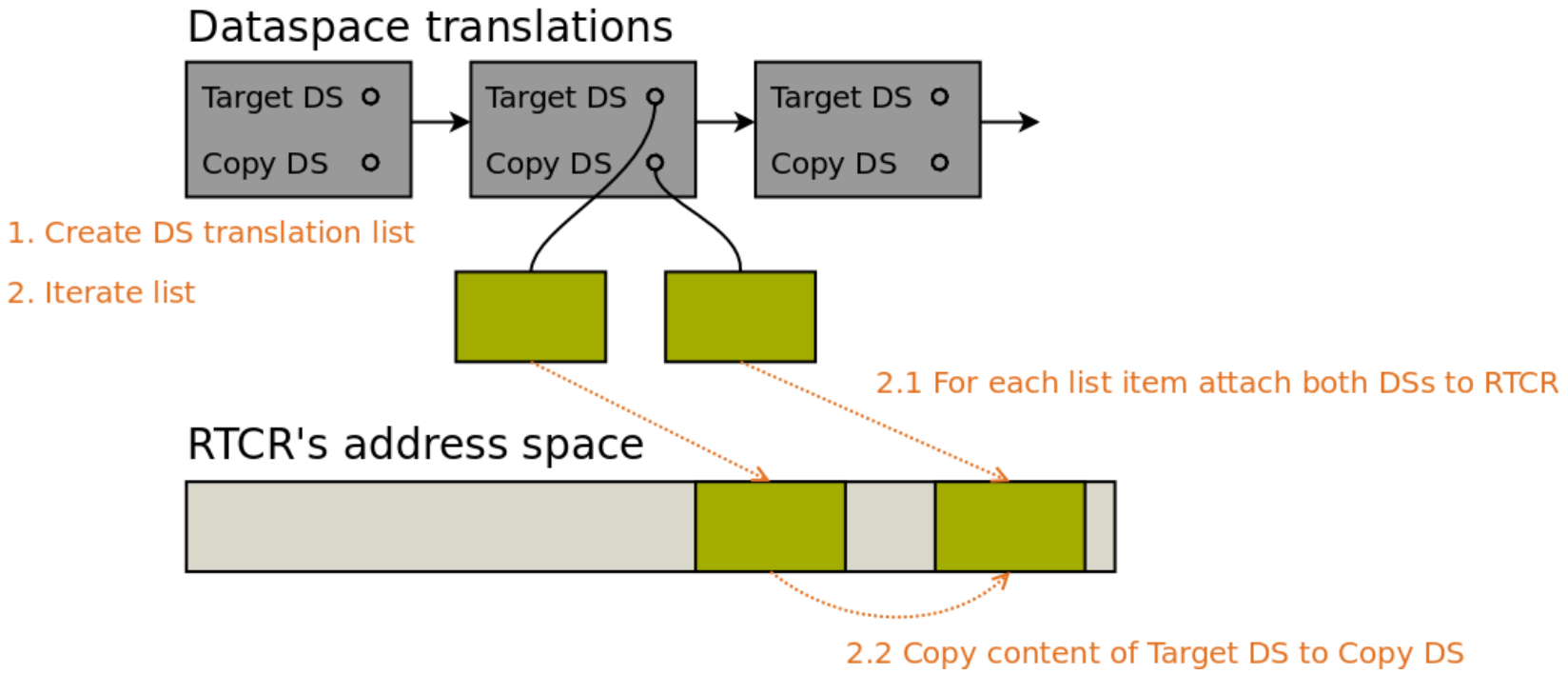
3. RTCR: Online/Offline Storage

- **Online storage**
  - Saves target state (session info) during runtime
  - Consists of mutable data structures
- **Offline storage**
  - Transformed version of online storage
  - Created during checkpoint process
  - Immutable
  - Represents the snapshot

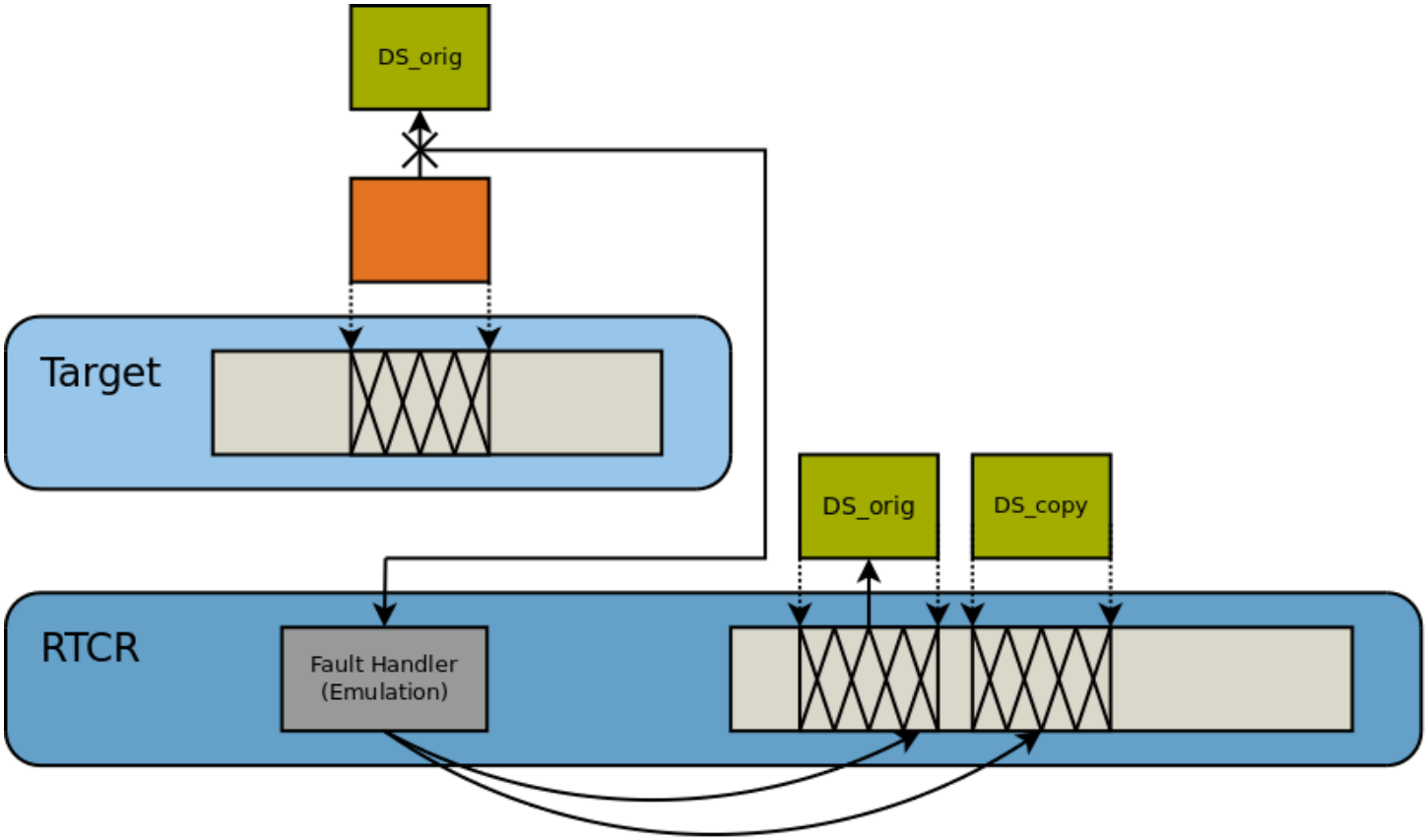
3. RTCR: Checkpointing Procedure

1. **Storing state of the capability space**
   - Identify location of capability space in target memory
   - Saving information about capability space layout
2. **Copying information from online to offline storage**
   - Access online storage structures for each intercepted service
   - Iterate over online storage structures
   - Copy all information to corresponding offline storage structure
3. **Copying Memory contents (depending on RTCR variant)**
   - SM: detection of target‘s dataspaces + explicit copying
   - RM: no copying needed (implicitly created copy already exists)
   - Inc: Identification of modified memory areas + explicit copying

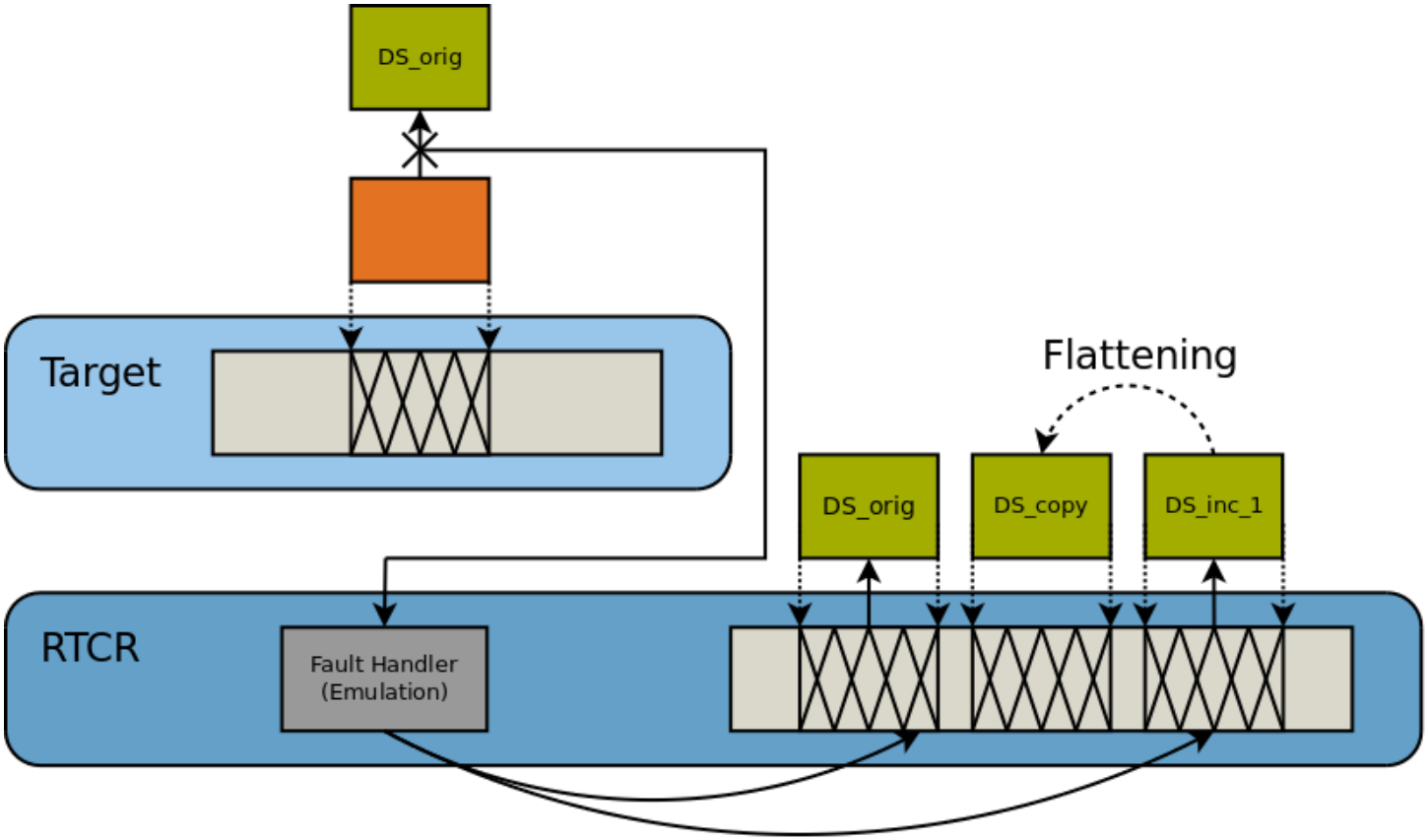
3. Shared Memory Checkpointing



3. Redundant Memory Checkpointing - Emulation



3. Redundant Memory Checkpointing - Flattening

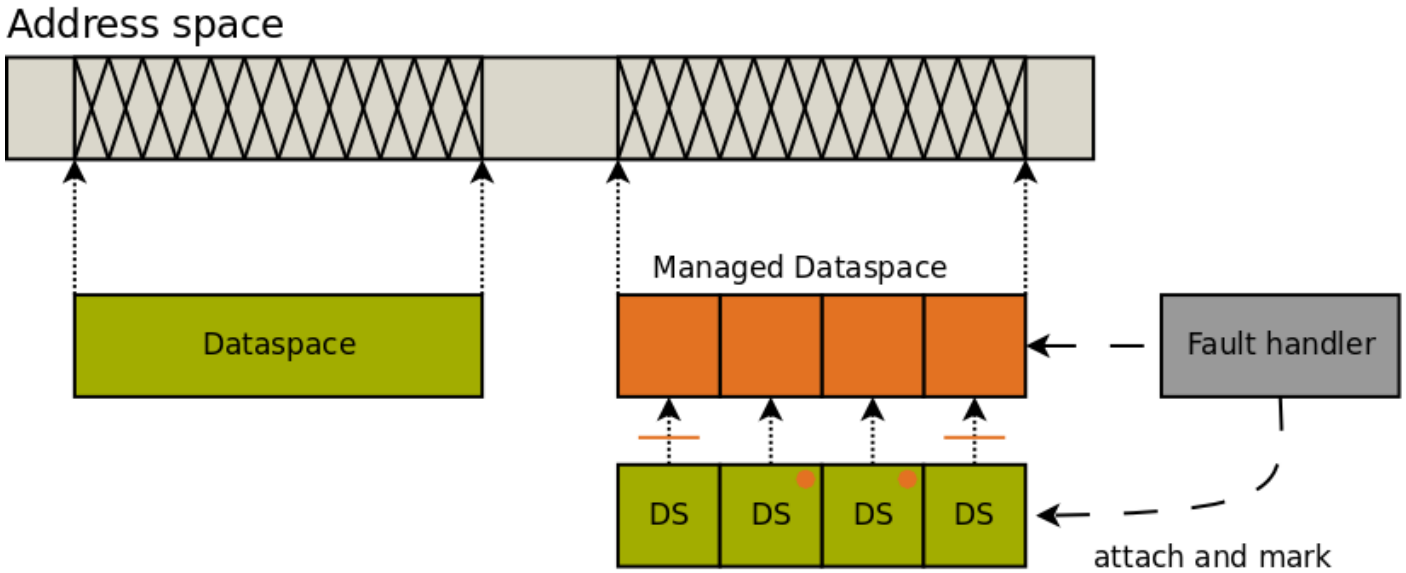


3. Incremental Memory Checkpointing

Idea: Only modified memory regions are checkpointed

→ Less amount of data to copy

→ Shorter stop time



3. RTCR: Other Software-based Optimizations

- Read-only memory (ROM)
- Copy-on-write (COW)
- Parallelization (vertical & horizontal)

Still under development / testing!

4. Hardware-based Optimizations

Purely Software-based variants of RTCR:

- Naive implementation
- RM seems a bit hacky

Solution:

- Development of HW accelerators that provide the similar functionality
- Usage of FPGAs

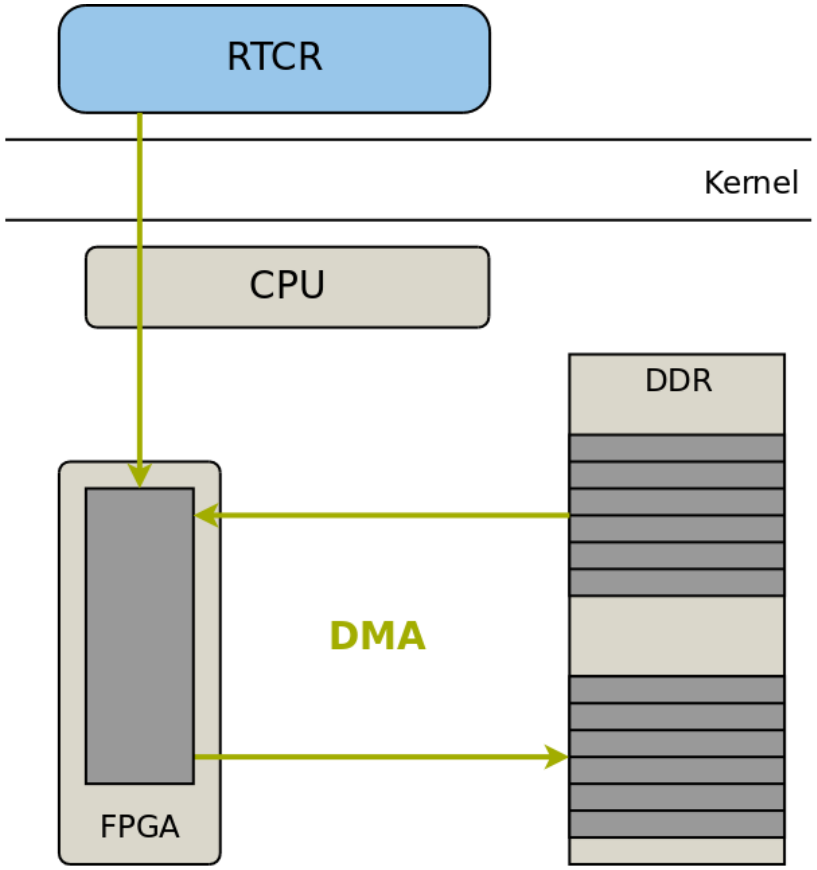
4. HW-based Opt: Custom Co-Processor

Development of a custom co-processor to accelerate memory copying

- FPGA-based (Xilinx Zynq-7000)
- DMA approach (AXI CDMA)

RTCR provides:

- Start addresses of memory area
- Size of both memory areas



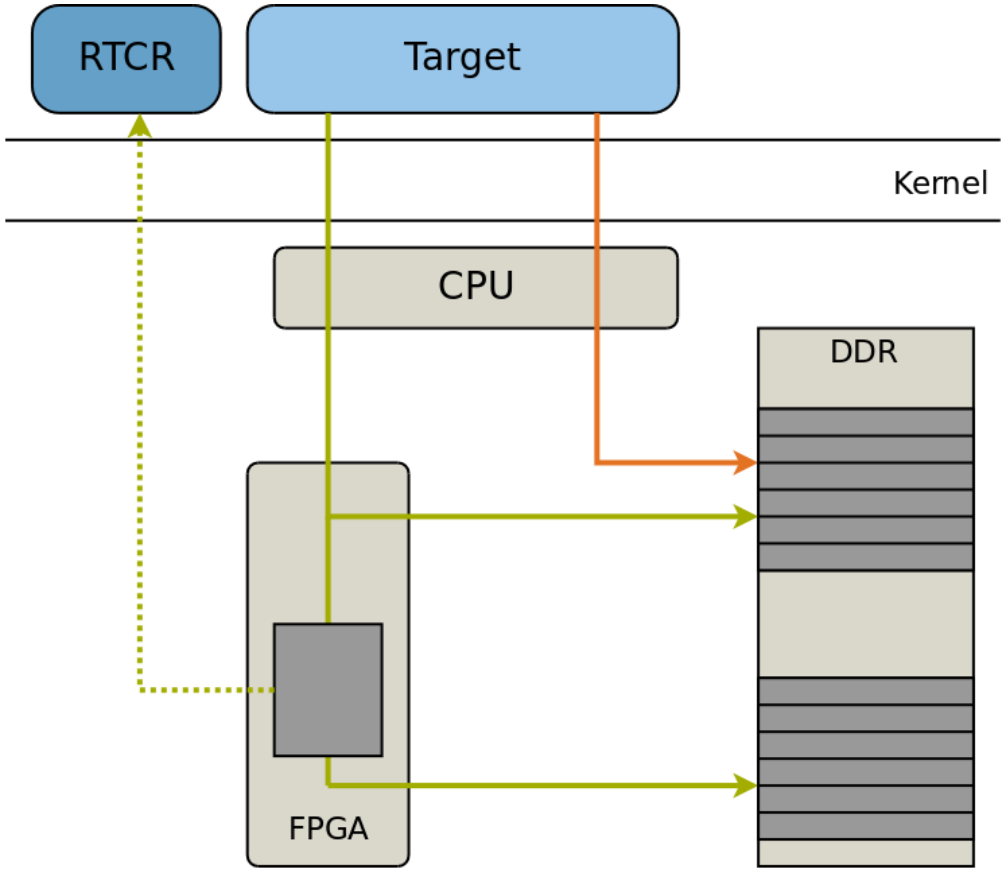
4. HW-based Opt: Custom Memory Interceptor

Custom interceptor component for efficient memory tracing and copying

- FPGA-based (Xilinx Zynq-7000)
- Redundant memory copying in HW

RTCR provides:

- Start addresses of memory area
- Size of both memory areas



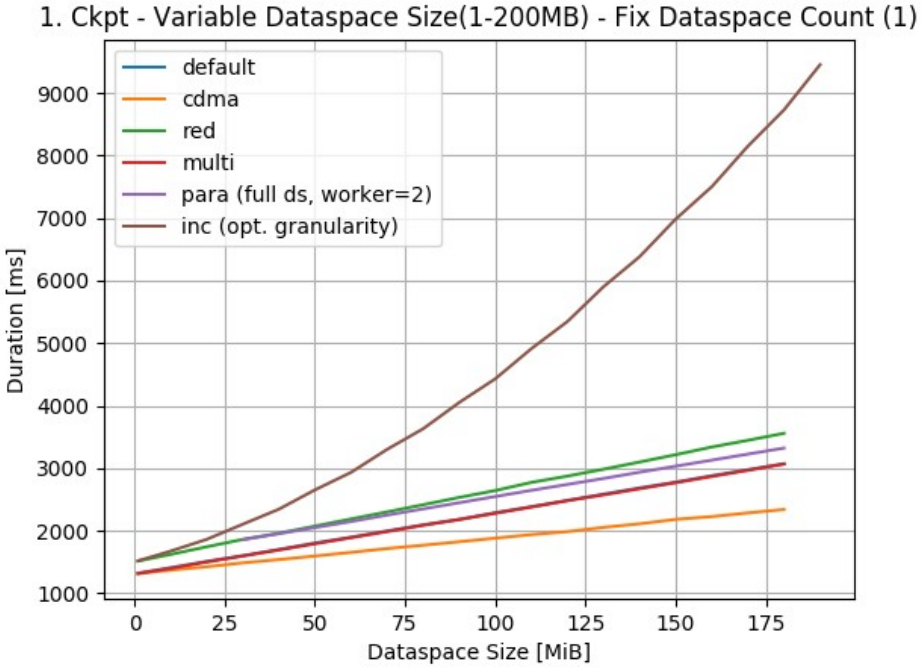
5. Evaluation

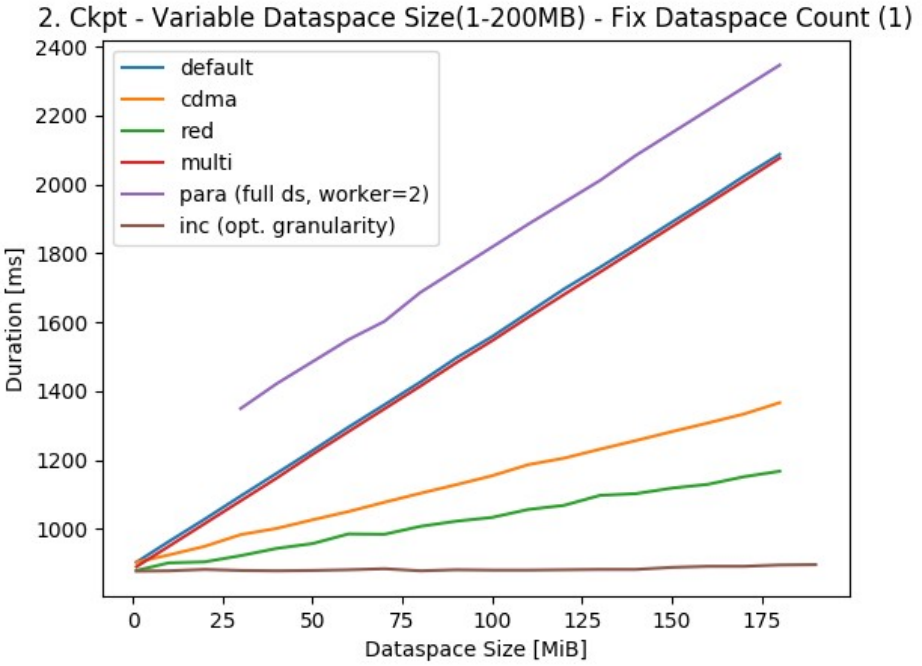
Memory checkpointing is the bottleneck!

Test Cases:

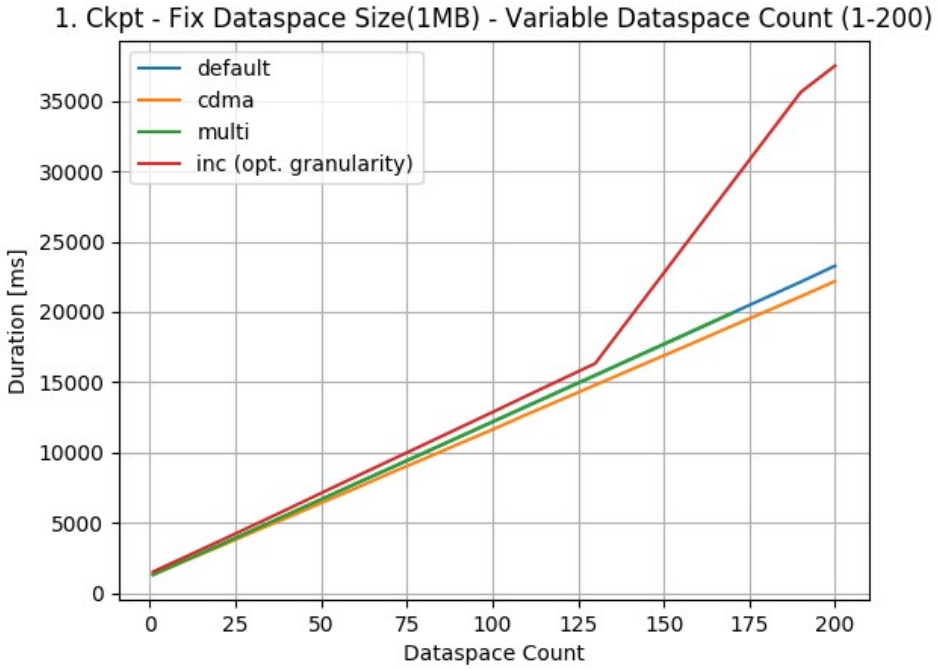
- Impact of amount of memory used by the component
- Impact of number of dataspaces
- Impact of granularity for incremental checkpointing

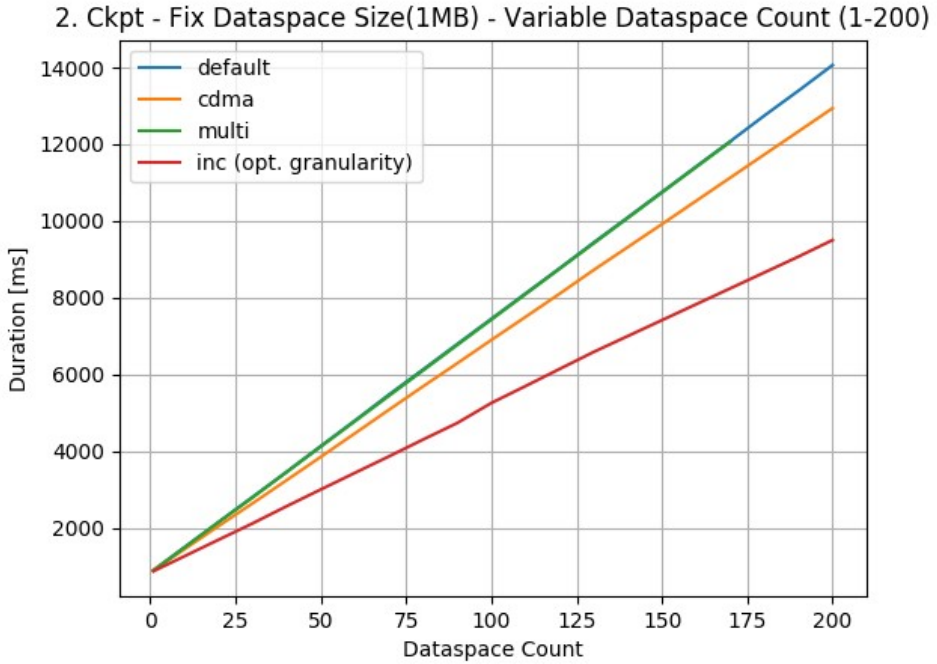
5. Evaluation: Impact of Dataspace Size



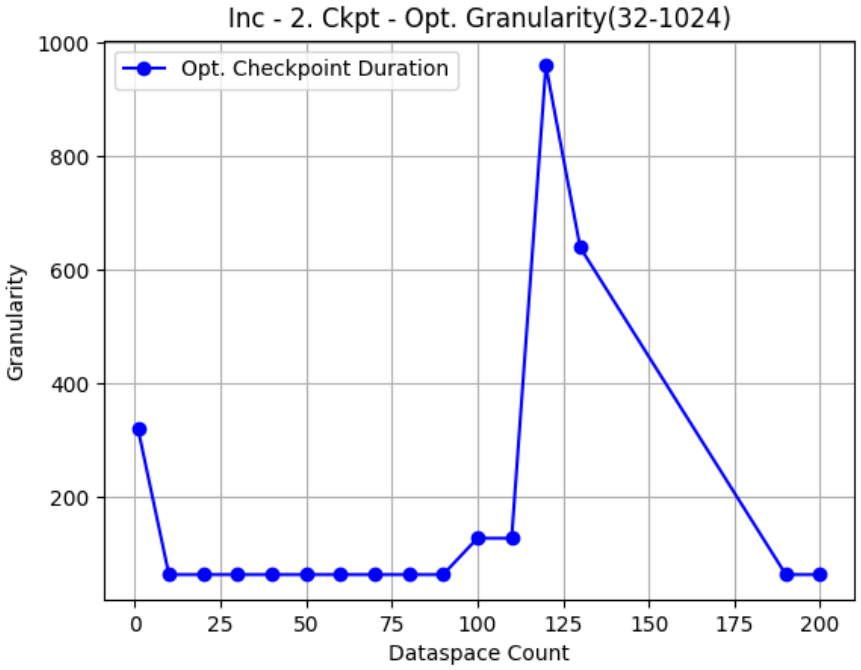


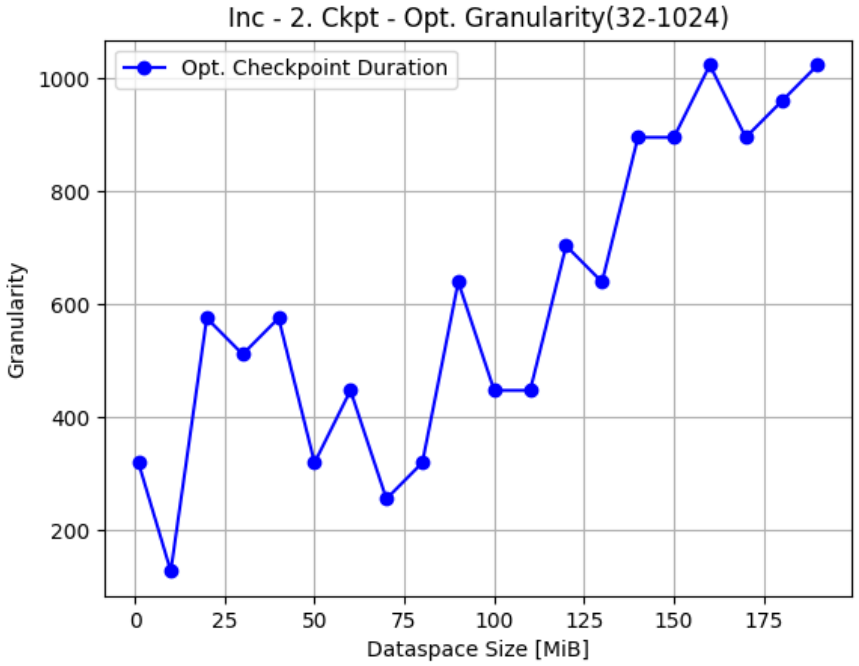
5. Evaluation: Impact of Dataspace Count





5. Evaluation: Optimal Granularity





6. Ongoing Work

- **Parallelization**
  - Adapting RTCR to multithreading
  - Experimenting with multiple instances of RTCR running simultaneously
- **HW optimizations**
  - Improving HW interceptor: AXI-Lite as proof of concept
    → AXI-Stream / AXI-Full for speedup
  - Checkpointing acceleration via MicroBlaze SoftCore processor
  - CPU modification based on RISC-V
- **Porting of RTCR to seL4/Genode**
- **Fixing existing restorer problems**
- **Testing & Evaluation**

7. Conclusion

- Solid C/R foundation for L4 microkernel based OS
- Solely SW based C/R might not be fast enough on current COTS HW
- Development of dedicated components to support & accelerate specific aspects of C/R in HW

Sources

[1] https://genode.org/documentation/architecture/index