Mapping Unikernels with TAG based architectures

Akilan Selvacoumar

Mathematics and Computer Sciences
Heriot Watt University

Year 1 progression report of:
Doctor of Philosophy

October 2022

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Akilan Selvacoumar
October 2022

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Nomenclature

Roman Symbols

F	complex function

Greek Symbols

γ	a simply closed curve on a complex plane
$\imath$	unit imaginary number $\sqrt{-1}$
π	$\simeq 3.14\ldots$

Superscripts

j	superscript index

Subscripts

0	subscript index
crit	Critical state

Other Symbols

$\oint_\gamma$	integration around a curve γ

Acronyms / Abbreviations

ALU	Arithmetic Logic Unit
BEM	Boundary Element Method
CD	Contact Dynamics
CFD	Computational Fluid Dynamics

CIF Cauchy's Integral Formula

CK Carman - Kozeny

DEM Discrete Element Method

DKT Draft Kiss Tumble

DNS Direct Numerical Simulation

EFG Element-Free Galerkin

FEM Finite Element Method

FLOP Floating Point Operations

FPU Floating Point Unit

FVM Finite Volume Method

GPU Graphics Processing Unit

LBM Lattice Boltzmann Method

LES Large Eddy Simulation

MPM Material Point Method

MRT Multi-Relaxation Time

PCI Peripheral Component Interconnect

PFEM Particle Finite Element Method

PIC Particle-in-cell

PPC Particles per cell

RVE Representative Elemental Volume

SH Savage Hutter

SM Streaming Multiprocessors

USF Update Stress First

USL Update Stress Last

Chapter 1

Introduction

Chapter 2

Motivation

Chapter 3

Research Questions

Chapter 4

Literature Review

The literature review is split into 3 sections. The first section talks about the papers surveyed for Unikernels and the 2nd section talks about papers surveyed for TAG based architectures and the third sections talks about the possible incentives of combining them both which helps answer the research questions stated (TODO: Add reference to research question section).

4.1 TAG based architecture survey

The following was a survey conducted on exisisting TAG based implementations and the recent survey based on TAG based architectures (//TODO add survey reference) published in 2022 was a good staring point to understand about various implementations of TAG based archtectures with the high level metrits and limitations. The following section provides our own version of the Survey to help decide the best implementations to answer the research questions (//TODO reference research questions chapter).

According to the TAG based architecture survey (//TODO add survey reference) there are 37 published efforts on TAG based architectures over the past decade and 20 published efforts preceding that.

4.1.1 Timder V

It is a tagged memory architecture for flexible and efficient isolation of code and data on small embedded systems. The TAG isolation is augmented with a memory protection unit to isolate induvidual processes. Timber V is compatible with exsisting code. The contributions of the paper are:

- Efficient tagged memory architecture for isolated execution on low-end processors.

- Concept introducted called stack interleaving that allows efficient and dynamic memory management.
- Lightweight shared memory between enclaves.
- Efficient shared MPU (i.e Memory Protection Unit) design.

4.1.2 ARM MTE

The ARMv8.5-Memory Tagging Extension (MTE) aims to increase the memory safety written for unsafe languages without requiring source code changes and in certain cases without recompilation. It generally foicuses on the bounds checking use case, Though it provides limited tags which means it can only provide probablilistic overflow detection. It is one of the latest commercial incarnations of memory-safety-focused tagged architectures.

4.1.3 D-RI5CY

It provides a design a design and implementation of a hardware dynamic information flow tracking (DIFT) architecture for RISC-V processor cores. The paper presents a low overhead implementation of DIFT that is specialized for low-end embedded systems for IOT applications. The following are high level contributions:

- Design f D-RI5CY, A DIFT-protected implementation of the RI5CY processor core. The paper implements the modification of the DIFT TAG propogation and TAG checking mechanism in a way that is transparent to the execution of the regular instructions.
- Concept introducted called stack interleaving that allows efficient and dynamic memory management.
- Lightweight shared memory between enclaves.
- Efficient shared MPU (i.e Memory Protection Unit) design.

4.1.4 TMDFI

4.1.5 HyperFlow

It is a design and security implementation that offers security assurance because it is implemented using a security-typed hardware description language. It allows complex information

flow policies to be configured at run time. The paper introduces ChiselFlow, a new secure hardware description language. The contribution of the paper includes:

- Processor architecture and implementation designed for timing-safe information flow security.
- Complere RISC-V intruction set extended with intructions for information flow control.
- Verified at design time with a hardware description language.
- Novel representations of lattices that can be implemented in hardware efficiently.

Hyperlow implements a nonmalleable IFC policy using tags. To eliminate timing side channels, the processor tracks the tag of the currently executing code and lushes caches, TLB, branch predictor, and other microarchitectural state on changes in the conidentiality or integrity tag of the running code. The modifications to avoid timing side channels seem more extensive than those to add tags. The authors report overheads in cycles per instruction of between 1operation to the worst-case number of cycles.

4.1.6 SDMP

This paper focuses on designing metadata tag based stack-protection security policies for general purpose tagged architecture. The policies specifically exploit the natural locality of dynamic program call graphs to achieve cacheability of the metadata rules that they require. The simple Return Address Protection policy has a performance overhead of 1.2% but just protects return addresses. The two richer policies present, Static Authorities and Depth Isolation, provide object-level protection for all stack objects. When enforcing memory safety, The Static Authorities policy has a performance overhead of 5.7% and our Depth Isolation policy has a performance overhead of 4.5%. The contribution of the paper includes:

- The formulation of a range of stack protection policies within the SDMP model.
- Three optimizations for our stack policies: Lazy Tagging, Lazy Clearing and Cache Line Tagging.
- The performance modeling results of our policies on a standard benchmark set, including the impact of our proposed optimizations.

4.1.7 Typed Architecture

This paper introduces Typed Architectures, a high-efficiency, low-cost execution sub- strate for dynamic scripting languages, where each data variable retains high-level type information at an ISA level. Typed Architectures calculate and check the dynamic type of each variable implicitly in hardware, rather than explicitly in software. Typed Architectures provide hardware support for flexible yet efficient type tag extraction and insertion, capturing common data layout patterns of tag- value pairs. The evaluation using a fully synthesizable RISC-V RTL design on FPGA shows that Typed Architectures achieve mean speedups of 11.2% and 9.9% with imum speedups of 32.6grade scripting engines for JavaScript and Lua. The contribution of the paper includes:

- ISA extension to efficiently manage type tags in hardware, which can be flexibly applied to multiple scripting languages and engines.
- Design and implement the Typed Architecture pipeline, which effectively reduces the overhead of dynamic type checking at low hardware cost.
- Prototype the proposed processor architecture us- ing a fully synthesizable RTL model to execute two production-grade scripting engines with large inputs on FPGA (executing over 274 billion instructions in total) and provide a more accurate estimate of area and power using a TSMC 40nm standard cell library.

4.1.8 Dover

4.1.9 Shakti-T

4.1.10 HDFI

4.1.11 lowRISC

4.1.12 Taxi

4.1.13 Pump

4.1.14 CHERI

4.1.15 SPARC M7/M8 SSM

4.1.16 Low-Fat Pointers

4.1.17 SAFE

4.1.18 DataSafe

4.1.19 Harmoni

4.1.20 Shioya, et al.

4.1.21 SIFT

4.1.22 FlexCore

4.1.23 Execution Leases

4.1.24 GLIFT

4.1.25 TIARA

4.1.26 DIFT Coprocessor

4.1.27 HardBound

4.1.28 Loki

4.1.29 FLexiTaint

4.1.30 SECTAG

4.1.31 Raksha

4.1.32 SecureBit

Chapter 5

Expirements

Chapter 6

Research Goals

Chapter 7

Research Timeline

Chapter 8

Conclusion

References

References

Index