Packet Processing Algorithm Identification using Program Embeddings

S. VenkataKeerthy, Yashas Andaluri, Sayan Dey, Rinku Shah, Praveen Tammana, Ramakrishna Upadrasta

2nd July, 2022
6th Asia-Pacific Workshop on Networking (APNET)
Introduction: Overall theme

**Fact:** CPU packet-processing speeds « network speeds

**SmartNICs:** High speed programmable hardware for packet processing

Control plane

- **CPU**
  - High CPU load
  - High packet communication

Data plane

- **NIC**
  - Fixed functionality

- **CPU**
  - Low CPU load
  - Low packet communication

- **SmartNIC**
  - Programmable, More packet processing features, Accelerators

- **ASIC/FPGA/SOC**
SmartNICs

- Generic cores for packet processing specified by Network Function (NF) program

Significant performance improvements
- Latency, throughput and power

IPipe\(^1\): IPSec program achieved 22.9 Gbps on 25GbE SmartNIC

\[\text{IPipe}\(^1\): Offloading Distributed Applications onto SmartNICs Using IPipe: Ming Liu, Tianyi Cui, Henry Schuh, Arvind Krishnamurthy, Simon Peter, and Karan Gupta (ACM Special Interest Group on Data Communication 2019)\]
SmartNICs

- Hardwired logic for frequently used operations and algorithms

**Goal:** Offloading NF programs from CPUs to SmartNIC accelerators
Accelerators available on SmartNICs

- Netronome Agilio CX
- Nvidia Bluefield 3 DPU
- Cavium LiquidIO
- Marvell OCTEON 10
- Pensando DSC-100

### Algorithms with Accelerator Support

<table>
<thead>
<tr>
<th></th>
<th>Hash</th>
<th>AES</th>
<th>RSA</th>
<th>IPSec</th>
<th>LPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netronome Agilio CX</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Nvidia Bluefield 3 DPU</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cavium LiquidIO</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Marvell OCTEON 10</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Pensando DSC-100</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Network Functions and Associated Algorithms

- L3 router
- Stateful firewall
- VPN gateway
- Intrusion Detection System
- 5G network functions
- Distributed data stores

<table>
<thead>
<tr>
<th>Algorithms used</th>
<th>Hash</th>
<th>AES</th>
<th>RSA</th>
<th>IPSec</th>
<th>LPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3 router</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Stateful firewall</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>VPN gateway</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Intrusion Detection System</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5G network functions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Distributed data stores</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
## Network Functions and Associated Algorithms

**Network Function program**

<table>
<thead>
<tr>
<th>Network Function program</th>
<th>Algorithms used</th>
</tr>
</thead>
<tbody>
<tr>
<td>L3 router</td>
<td>Hash</td>
</tr>
<tr>
<td>Stateful firewall</td>
<td>![checkmark]</td>
</tr>
<tr>
<td>VPN gateway</td>
<td>![checkmark]</td>
</tr>
<tr>
<td>Intrusion Detection System</td>
<td>![checkmark]</td>
</tr>
<tr>
<td>5G network functions</td>
<td>![checkmark]</td>
</tr>
<tr>
<td>Distributed data stores</td>
<td>![checkmark]</td>
</tr>
</tbody>
</table>

**NF Programs** → **SmartNIC Accelerators**
Difficulties in Mapping

- Identification of regions of code suitable for accelerators
  - Same algorithm can be implemented in multiple ways

- Porting to SmartNICs needs analysis and multiple rounds of manual tuning
  - Tune program by utilizing SmartNIC accelerators

Identifying + Mapping NF to SmartNIC is a tedious and laborious process

Can this process be simplified?
Problem Statement

- Need a workflow to simplify the cross-platform porting process
- Automatic identification of regions in Network Functions

Network Function

AES_Encrypt(){
  ...
}

CRC(){
  ...
}

Accelerators

<table>
<thead>
<tr>
<th>Crypto</th>
<th>Load Balancer</th>
<th>Atomic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Look-up</td>
<td>Queue</td>
<td>Bulk</td>
</tr>
<tr>
<td>Statistics</td>
<td>CAM</td>
<td>Hash</td>
</tr>
</tbody>
</table>
Approach

Our view: This is a ML classification problem.

Our approach:

- **Use Compilers** to aid developers to map NF program to SmartNIC
- **Use ML** to identify code regions performing a specific task (algorithm)
- **Create realistic dataset** of packet processing algorithms
Why ML?

- **Undecidability**
  - It is hard to identify algorithms in a program

- **Laborious**
  - Manually assigning accelerators for functions in a large NF program is tedious

- **Scale of variation**
  - Diverse algorithms and SmartNIC architectures
Challenges

1. **Represent** algorithms and programs as input to ML model

2. **Create dataset** of packet processing algorithms
   - Realistic
   - Diverse
   - Wide range of applicability
Challenge #1: Representations of Programs
Background: LLVM IR

**LLVM IR**: LLVM IR is the Intermediate Representation (IR) of the LLVM compiler toolchain.

### C

```c
int sum(int a, int b)
{
    return (a+b);
}
```

### Front-end

```assembly
define i32 @sum(i32 %a, i32 %b) #0 {
    entry:
        %a.addr = alloca i32, align 4
        %b.addr = alloca i32, align 4
        store i32 %a, i32* %a.addr, align 4
        store i32 %b, i32* %b.addr, align 4
        %0 = load i32, i32* %a.addr, align 4
        %1 = load i32, i32* %b.addr, align 4
        %add = add nsw i32 %0, %1
        ret i32 %add
    }
```
### Background: Program Representations

Various techniques in use

| Collecting **features** using domain expertise | Specific task (Domain expertise) |
| Programs as **tokens** of natural languages | Syntactic |
| **Abstract Syntax Tree** representations | Syntactic + Limited Semantic |
| **IR-based** representations | Syntactic + Semantic + Generalized |
Background: IR2Vec: IR based Program Embeddings

Proposed Methodology

- Identify appropriate accelerators for the program
  - Use ML based techniques

- Utilize IR2Vec embeddings
  - Encodes syntactic and semantic information of the program

- Predict accelerator label for each function
Proposed Methodology (contd.)

Source code

```c
void AES_Encrypt()
{
    int k = 2;
    ...
}

void AES_CBC_Encrypt()
{
    ...
    AES_Encrypt()
    ...
}
```

LLVM IR

```c
define void @AES_Encrypt()
{
    %k = alloca i32
    ...
}
```

Callee: AES_Encrypt()

```c
define void @AES_CBC_Encrypt()
{
    ...
    call AES_Encrypt()
    ...
}
```

Caller: AES_CBC_Encrypt()

```c
VAR = alloca INT
...
```

ML classifier

IR2Vec Module
Challenge #2: Generation of Dataset
Dataset Creation

- ML needs more data
  - ImageNet - 14 million images
  - COCO - 330K images

- Lack of availability of sufficient real world NF programs
  - Earlier datasets [Clara]: only around 7.5k programs

- Need to create a custom dataset
  - using programs from NF domain

*Clara: Automated SmartNIC Offloading Insights for Network Functions: Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, Ang Chen (ACM Symposium on Operating Systems Principles 2021)*
### Initial Steps: Seed Dataset Collection

**Seed dataset:** Collected functions for algorithms used in cryptography libraries

<table>
<thead>
<tr>
<th>Library</th>
<th>Algorithm</th>
<th>OpenSSL (v1.1)</th>
<th>OpenSSL (v3)</th>
<th>CryptoPP (v8.6)</th>
<th>Botan (v2.19)</th>
<th>Nettle (v3.7)</th>
<th>WolfCrypt (v5.1)</th>
<th>MbedTLS (v3.1)</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td></td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>42</td>
</tr>
<tr>
<td>DES</td>
<td></td>
<td>13</td>
<td>13</td>
<td>8</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td>RSA</td>
<td></td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>31</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>26</td>
<td>27</td>
<td>20</td>
<td>15</td>
<td>6</td>
<td>15</td>
<td>12</td>
<td>121</td>
</tr>
</tbody>
</table>
Dataset Expansion using Compiler Transformations

**Method:** Apply compiler transformations on original (seed) NF programs

- Adds diversity to dataset
  - Code size of the program
  - Latency
  - Throughput
  - Power usage

- Semantics of original codes are preserved

**Result:** Produces sufficient data for training a ML model
Dataset Expansion using Compiler Transformations

- Apply random permutations of LLVM transformations to programs

300 permutations

-const-prop -dce ... -loop-simplify -instcombine

Original IR

AES_encrypt(){
  %1 = alloca i32
  ...
}

Standard optimization sequences

-O1, -O2, -O3, -Os, -Oz

Processed dataset ~ 37K programs

305 Transformed equivalent programs

p1, pi, pn, pstd
Experimentation & Implementation details

Experimentation

- Detection of CRC algorithm
  - Classifying CRC and non-CRC programs
- Detection of cryptography algorithms
  - CRC, AES, DES/3DES, RSA, non-NF programs

Implementation

- Manually labelled functions for classification
- Used IR2Vec embeddings of programs compiled to LLVM IR (v12.0)
- Compared results from our approach with Clara

Clara: Automated SmartNIC Offloading Insights for Network Functions: Yiming Qiu, Jiarong Xing, Kuo-Feng Hsu, Qiao Kang, Ming Liu, Srinivas Narayana, Ang Chen (ACM Symposium on Operating Systems Principles 2021)
## Results: Precision

<table>
<thead>
<tr>
<th>Model</th>
<th>CRC</th>
<th>CRC + Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clara</td>
<td>IR2Vec</td>
</tr>
<tr>
<td>Gradient Boosted Decision Tree</td>
<td>0.992</td>
<td>0.974</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.983</td>
<td>0.994</td>
</tr>
<tr>
<td>Multi-layer Perceptron</td>
<td>0.983</td>
<td>0.997</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>0.992</td>
<td>0.999</td>
</tr>
<tr>
<td>k-Nearest Neighbour</td>
<td>0.980</td>
<td>0.999</td>
</tr>
<tr>
<td>AutoML</td>
<td>0.980</td>
<td>0.999</td>
</tr>
</tbody>
</table>
## Results: Recall

<table>
<thead>
<tr>
<th>Model</th>
<th>CRC</th>
<th></th>
<th>CRC + Cryptography</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clara</td>
<td>IR2Vec</td>
<td>Clara</td>
<td>IR2Vec</td>
</tr>
<tr>
<td>Gradient Boosted Decision Tree</td>
<td>0.435</td>
<td>0.997</td>
<td>0.594</td>
<td>0.947</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.488</td>
<td>0.995</td>
<td>0.622</td>
<td>0.968</td>
</tr>
<tr>
<td>Multi-layer Perceptron</td>
<td>0.437</td>
<td>0.999</td>
<td>0.590</td>
<td>0.958</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>0.484</td>
<td>0.995</td>
<td>0.604</td>
<td>0.894</td>
</tr>
<tr>
<td>k-Nearest Neighbour</td>
<td>0.486</td>
<td>0.999</td>
<td>0.630</td>
<td>0.974</td>
</tr>
<tr>
<td>AutoML</td>
<td>0.487</td>
<td>0.999</td>
<td>0.621</td>
<td>0.978</td>
</tr>
</tbody>
</table>
### Results: F1 Score

<table>
<thead>
<tr>
<th>Model</th>
<th>CRC</th>
<th>CRC + Cryptography</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Clara</td>
<td>IR2Vec</td>
</tr>
<tr>
<td>Gradient Boosted Decision Tree</td>
<td>0.605</td>
<td>0.985</td>
</tr>
<tr>
<td>Decision Tree</td>
<td>0.652</td>
<td>0.994</td>
</tr>
<tr>
<td>Multi-layer Perceptron</td>
<td>0.605</td>
<td>0.998</td>
</tr>
<tr>
<td>Support Vector Machine</td>
<td>0.651</td>
<td>0.997</td>
</tr>
<tr>
<td>k-Nearest Neighbour</td>
<td>0.650</td>
<td>0.999</td>
</tr>
<tr>
<td>AutoML</td>
<td>0.651</td>
<td>0.999</td>
</tr>
</tbody>
</table>

IR2Vec can capture semantics of the algorithm
Summary & Future Work

Contributions

- Using embedding techniques (IR2Vec) to represent programs from network domain
- Modeling algorithm identification problem with a scalable ML approach
- Realistic dataset collection and generation of semantically equivalent programs

Future Work

- Applying to real-world network functions
- Identifying other algorithms
Authors

S. VenkataKeerthy  
PhD student, IITH

Yashas Andaluri  
MTech student, IITH

Sayan Dey  
MTech student, IITH

Rinku Shah  
Asst. Professor, IIITD

Praveen Tammana  
Asst. Professor, IITH

Ramakrishna Upadrasta  
Asst. Professor, IITH