# FemtoCOF: A FemtoCloud Offloading Framework With a Machine-Dependent Optimizer

Laila Elbeheiry Mayssa Zaki

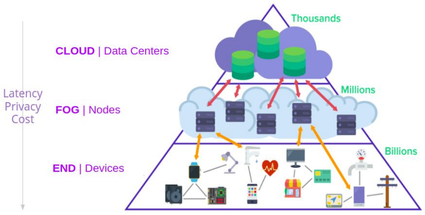
Advisors: Khaled Harras, Giselle Reis

## I. INTRODUCTION

### A) Motivation

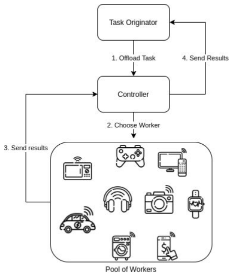
- The Internet of Things industry is rapidly expanding; 79.4 zettabytes (1 followed by 21 zeros) of data are due to arrive by 2025. [1]
- IoT devices cannot process this amount and must offload to the Cloud/Fog/Edge.
- Mobile devices are expanding in numbers and capacity.
- Mobile/IoT devices should be **considered as offload candidates for the more constrained IoT devices** to enhance performance, utilization and privacy.

### B) Our Vision



### C) Background

- **FemtoClouds:** systems that leverage the available compute capacity of a group of heterogeneous mobile and IoT devices to offer computing services. [2]
- **Task Originators:** Clients that need to offload computation
- **Controller:** The device that manages the FemtoCloud
- **Workers:** The IoT/mobile devices to which tasks are offloaded
- **Offloading Technique:** Mechanism by which code is offloaded to the Controller



## II. PROBLEM

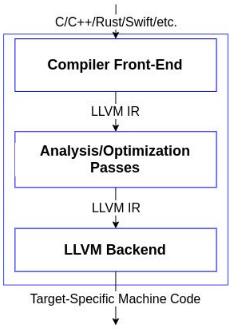
- FemtoClouds have distinguishing characteristics:
  - **Heterogeneous:** Huge variety in hardware architectures of Workers
  - **Footprint-Sensitive:** Offloaded task can't hinder the functionality of the Worker from the owner's perspective
  - **Dynamic:** Workers enter and exit the system at a high rate
- This leads to the following requirements on the offloading technique:
  - **Multi-Architecture Support:** Automatically run unmodified code on any hardware or Instruction Set Architecture
  - **Lightweightness:** Incur small footprint on resources
  - **Maximize Utilization:** Do not waste any compute cycles (to minimize footprint)
- Existing FemtoClouds and similar infrastructures do not provide an offloading technique that meets these requirements

## IV. OPTIMIZATION

### A) Optimizer Overview

- Main Contribution of FemtoCOF
- Given: **target specification**
- Leveraged **LLVM** as a cross-compiler
- Applies **Machine-Dependent Optimizations**
- *(WIP)* Outputs **machine-specific binaries**
- Based on an Optimizer called **Polly**

### B) LLVM Infrastructure

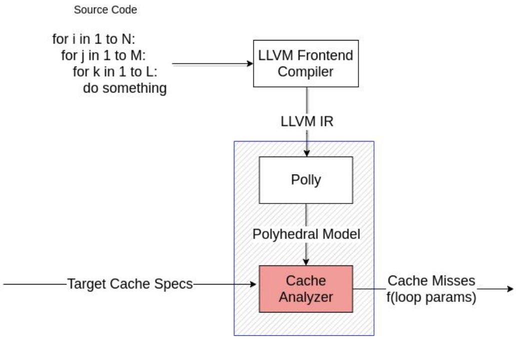


### C) Polly Optimizer

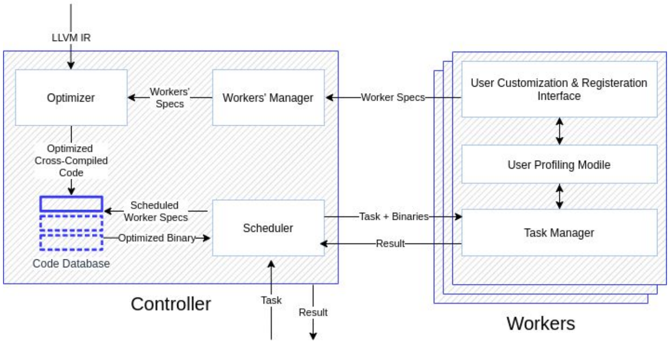
- Embedded inside the LLVM Compiler
- Transforms LLVM IR to a mathematical model called the **Polyhedral model** [3]
- Applies optimizations on that model
- Transforms the model back to LLVM IR

### D) Contribution: Cache Analyzer

- Given: Polyhedral model extracted by **Polly** + cache specs of a **Worker**
- Calculates the number of cache misses incurred by the program as a function of the loop parameters
- *(WIP)* Chooses the loop parameters that minimize the number of cache misses
- *(WIP)* Uses Polly to transform the loop using the calculated parameters



## III. SYSTEM ARCHITECTURE



### System Flow

**Phase 1 - Worker Registration:**

Each Worker sends its **specs** (architecture name, number of cache levels, each cache level parameters, number of cpus, floating-point units) to the Controller

**Phase 2 - App Registration:**

Task Originator sends app code to the Controller

Optimizer cross-compiles and optimizes the code for each target

Optimizer stores the binaries in a code database

**Phase 3 - Computation:**

Task Originator sends a request and data to the Controller

Controller chooses a Worker to send the corresponding precompiled app to

Worker runs the computation and sends back the results

## V. NEXT STEPS

### Implementation Plan

1. Integrate the cache analyzer within Polly
2. Transform the code based on cache analysis
3. Piece together Optimizer with other Controller modules to get a full-fledged offloading framework

### Evaluation Plan

**Benchmark Examples:**

- Matrix Multiplication
- Matrix Transpose and Vector Multiplication
- Gaussian Filter
- LU decomposition
- 2-D Image processing

**Metrics:**

- Running Time (s) -> **Performance**
- Cache Miss Rate (%) -> **Utilization**
- Disk Usage (Kb) -> **Lightweightness**

**Workers:**

- Raspberry Pi 3, Intel Edison, PC

## VI. CONCLUSION

- Designed FemtoCOF: an offloading framework designed from the ground up with FemtoCloud attributes in mind
- Implemented a cache analyzer based on the Polyhedral model
- *(WIP)* Integrated the cache analyzer with Polly to tailor Polly's code transformations for a target hardware specification
- Leveraged state-of-the-art tools like LLVM and Polly

Carnegie Mellon University Qatar

MEETING OF THE Minds Undergraduate Research Symposium

[1] Vega M. (2020). Internet of Things Statistics, Facts & Predictions. in Review 42.
[2] H. K. Gedawy, K. Habak, K. Harras, and M. Hamdi. Ramos: A resource-aware multi-objective system for edge computing. IEEE Transactions on Mobile Computing
[3] https://polly.llvm.org/