DECOMPOSER: Functional Decomposition of Monolithic Applications to Heterogeneous Resources in Disaggregated Environments

Guilherme Mendes Vieira de Matos**
guilherme.matos@estudante.ufscar.br
Department of Computer Science, UFSCar
Sorocaba, SP, Brazil

Fábio Luciano Verdi
Department of Computer Science, UFSCar
Sorocaba, SP, Brazil
verdi@ufscar.com

Washington R. D. Silva
Department of Computer Science, UFSCar
Sorocaba, SP, Brazil
washingtonrds@estudante.ufscar.br

Andrew Williams
Ericsson Research
Stockholm, Sweden
ndrew.williams@ericsson.com

Abstract

This paper analyzes the performance of disaggregating monolithic applications across heterogeneous hardware. A monolithic program with convolution functions was tested on AMD, Intel servers, and a BlueField-2 DPU (ARM). After disaggregation into independent processes, it was run on the same server and over a local network. Results showed that disaggregation maintained similar performance to the monolithic version, with minimal impact despite network latency. We also observed improved cache utilization, particularly a reduction in L1 cache misses and L2 accesses. Disaggregation shows promise in optimizing memory usage. Future work will focus on refining memory analysis and using GPUs and FPGAs to improve computational efficiency.

ACM Reference Format:
Guilherme Mendes Vieira de Matos, Fábio Luciano Verdi, Washington R. D. Silva, and Andrew Williams. 2024. DECOMPOSER: Functional Decomposition of Monolithic Applications to Heterogeneous Resources in Disaggregated Environments. In *Proceedings of the CoNEXT Student Workshop 2024 (CoNEXT-SW '24), December 9–12, 2024, Los Angeles, CA, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3694812.3699930

1 Introduction

CPUs are hitting limits due to the slowdown of Dennard scaling and Moore's Law. To address this, accelerators like GPUs, FPGAs, DPUs, and TPUs are now prevalent in modern datacenters.

Datacenter disaggregation is also evolving, with resources like memory, processing, and networking working in separate pools to boost scalability and management. In these heterogeneous environments, we introduce Decomposer, a method to split monolithic applications into functional blocks, optimizing resource use across architectures.

*Both authors contributed equally to this research.

CoNEXT-SW '24, December 9–12, 2024, Los Angeles, CA, USA

ACM ISBN 979-8-4007-1255-5/24/12
https://doi.org/10.1145/3694812.3699930

2 Decomposer workflow

Current solutions enable monolithic applications in disaggregated environments [2], but do not handle heterogeneous resources. Our approach focuses on distributing native applications across such resources in disaggregated setups. Our method integrates dynamic and static analysis to boost energy efficiency and performance.

The workflow (Figure 1) starts by generating intermediate representations (IR) from an application. Dynamic and static analysis identify hotspots, energy consumption, and execution times. Based on this, the application is decomposed and compiled with a socket driver for communication. We discuss the tools and techniques used below.

2.1 Intermediate representation

Disaggregating monolithic applications requires understanding their pipeline, including memory operations, functions, and flow. Tools like LLVM [3] and MLIR generate IRs and Control Flow Graphs (CFGs) to extract functions and compile them for different targets. After generating the IR, functions are analyzed to determine their target hardware. The extracted functions are compiled into binaries, and a driver, based on the CFG, ensures proper execution and communication across hardware.

2.2 Application analysis

In this multi-processor setup, identifying functional decomposition strategies is complex. Dynamic analysis collects data like execution time and energy consumption per processor. Hotspots are extracted and executed on suitable hardware. An algorithm compiles binaries for each target, which are tested on respective servers. Profiling, via gprof [1], stores results for analysis.

2.3 Communication deployment

After function extraction, communication between functional blocks is key. We created a program to handle memory data, execute the extracted function, and enable socket-based communication. For each function, the name, required memory, and target data (e.g., IP and port) are analyzed from the IR. This program coordinates communication and memory exchange between blocks.

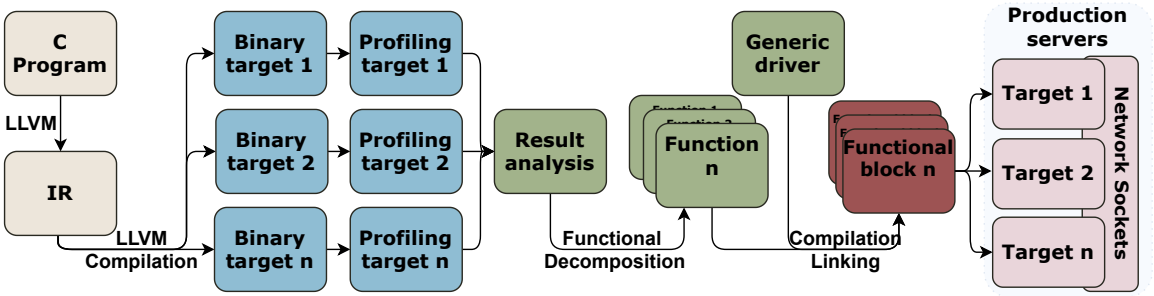

Figure 1: Decomposer workflow.

3 Implementation

We implemented a monolithic program with three convolution functions (conv1, conv2, conv3), performing 1, 2, and 4 hundred thousand iterations. The monolithic program was executed on three servers with AMD and Intel processors, and on a BlueField-2 DPU (ARM). Execution times were collected for performance analysis. We then disaggregated the program into four functions, running them as independent processes on the same servers, and collected new execution times. Hybrid setups were also tested, running the main function on BlueField-2 and the convolutions on AMD, and vice-versa, to assess performance the local network environments.

4 Results

Figure 2 shows the execution times for monolithic and disaggregated applications. In this simple evaluation, the disaggregated version on the same server maintained practically the same performance compared to the monolithic version. The disaggregated version on the network also remained in the same time range, even taking into account that the main function was being performed on an ARM and the existence of latency (in this case, around 1ms because it is a local network). However, Figure 3 shows better utilization of the cache memory, reducing L1 cache misses and L2 cache accesses.

5 Conclusions

This study investigated the performance of disaggregating monolithic applications across heterogeneous hardware. In our experiments, disaggregation on the same server showed negligible performance difference compared to the monolithic version, while disaggregation across a local network yielded similar results, despite higher network latency. Improved cache utilization, particularly in reducing L1 cache misses and L2 accesses, was observed. Future work will focus on further optimizing memory management and exploring hybrid environments involving GPUs and FPGAs to enhance computational performance in disaggregated setups.

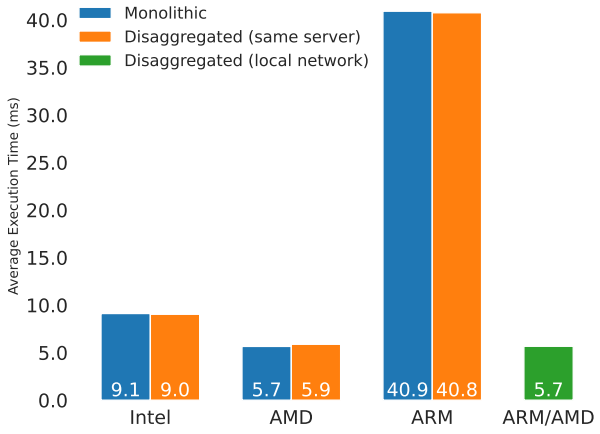

Figure 2: Execution time.

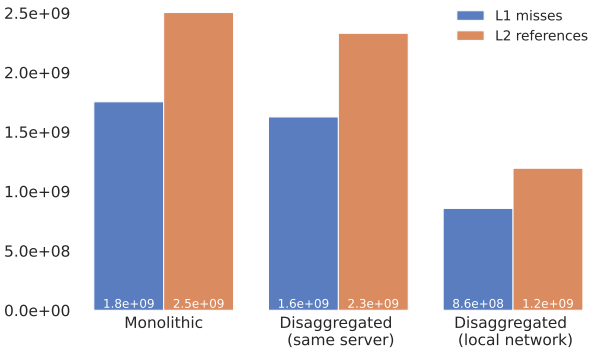

Figure 3: L1 Cache Misses and L2 Cache References (average).

Acknowledgment

This work was supported by Ericsson Telecomunicações Ltda., and by the Sao Paulo Research Foundation (FAPESP), 2021/00199-8, CPE SMARTNESS . This study was partially funded by *CAPES*, Brazil - Finance Code 001.

References

[1] Graham et al. 1982. Gprof: A call graph execution profiler. In *SIGPLAN* (Boston, Massachusetts, USA) *(SIGPLAN '82)*. New York, NY, USA, 120–126. https://doi.org/10.1145/800230.806987

[2] Guo et al. 2023. Mira: A Program-Behavior-Guided Far Memory System. In *SOSP '23* (Koblenz, Germany) *(SOSP '23)*. New York, NY, USA, 692–708. https://doi.org/10.1145/3600006.3613157

[3] Chris Lattner and Vikram Adve. 2004. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *CGO'04*. Palo Alto, California.