CADI: Semi-Automatic Insertion of Host-Device Data Transfer Directives Based on Code Analysis

Koki Watanabe[1] (watanabe.koki.t2@dc.tohoku.ac.jp), Masatoshi Kawai[1], Junji Fukuhara[2], Tomoyoshi Funazaki[2], Seiji Nishimura[2], Sichen Tao[1], Hiroyuki Takizawa[1]
[1]Tohoku University, [2]DENSO CORPORATION

Background & Goal

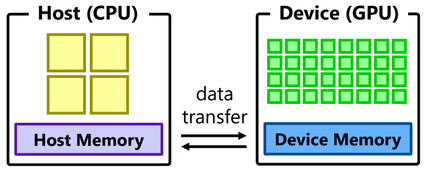
Widespread adoption of heterogeneous computing

- Computation is offloaded to devices such as GPUs
- Directive-based programming models such as OpenACC simplify GPU programming
- Heterogeneous systems may involve multiple memory spaces

➡ **Host-device data transfer is often required**

Data transfer methods

- **Manual data transfer directives**
  - Performance can be optimized
  - Increase programming effort and risk of errors
- **Automatic** (Unified Memory / Managed Memory)
  - Reduces programming effort by eliminating explicit data transfer directives
  - May introduce runtime overhead and has limited supported environments



Example of manual data transfer directives

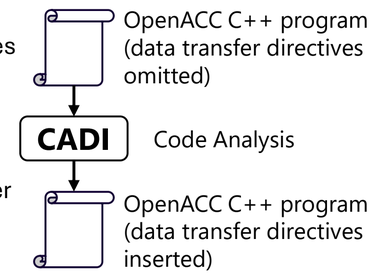
```
#pragma acc data copyout(p1[0:N])
```

Our Goal

- **Reduce programming effort while preserving opportunities for performance optimization**
  - Achieve a balance between automatic directive insertion and programmer-guided optimization

Approach

- **Semi-automatic** approach: **CADI** automates transfer-target identification, while programmers specify transfer directions
  - Targets OpenACC-based C++ programs where data transfer directives are omitted
  - Automatically inserts omitted data transfer directives through code analysis
  - Allows programmers to control performance-critical data movement



Related Work

DawnCC [1]

- Automatically inserts parallelization and data transfer directives using static code analysis
- Difficult to handle cases requiring runtime information, such as indirect memory accesses (e.g., x[y[i]])

CADI vs. DawnCC

- Combines static code analysis with runtime information
- Supports programs that are difficult to handle using static analysis alone

[1] Mendonca, Gleison and Guimaraes, B. P. M. G. F.: DawnCC: Automatic Annotation for Data Parallelism and Offloading, ACM Trans. Archit. Code Optim., Vol. 14, No. 2 (2017)

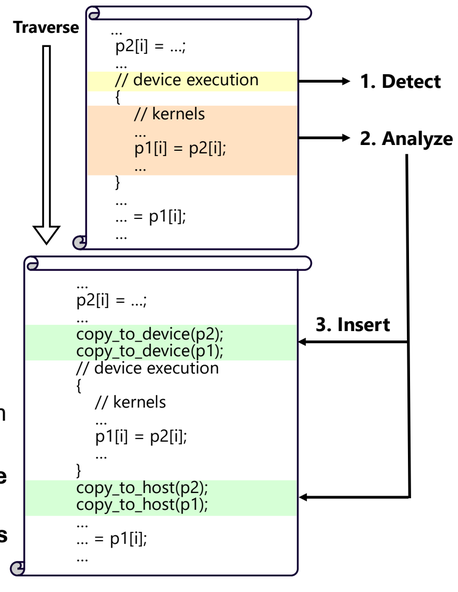
Proposed Method

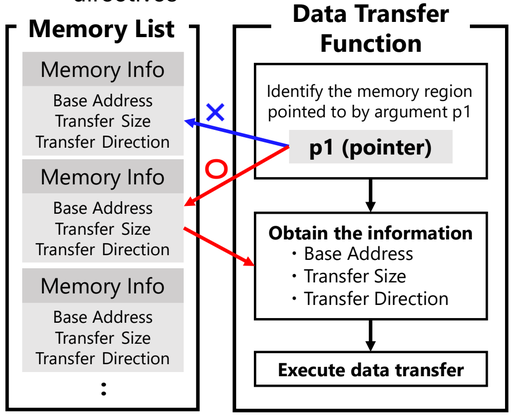
Overview

- Information required for data transfer
  - **Base address**
  - **Transfer size**
  - **Transfer direction**
- CADI automatically determines the base address and transfer size
- Transfer direction is specified by the programmer per allocated memory region

Code Analysis

- Detect the device execution region
- Analyze the region
  - **Extract pointers used by device computation**
- Insert **data transfer function calls**
  - Equivalent to data transfer directives





Runtime Memory Tracking

- Track allocated memory regions
- Data transfer functions
  - Identify target memory regions using runtime memory tracking
  - **Obtain the required transfer information**
  - Execute data transfer

Evaluation & Discussion

Evaluation Method

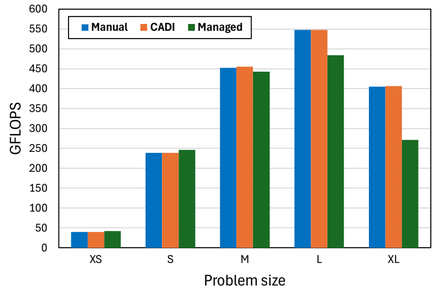
- Performance evaluation using the Himeno benchmark
- Comparison targets
  - **Manual implementation**
  - **CADI (proposed method)**
  - **Managed Memory**
- For Manual and CADI, transfer directions are manually optimized.
- Measure computational performance (GFLOPS)

Evaluation Results

- Execution performance comparable to manual implementation
  - **Performance degradation relative to the manual implementation is less than 1%**
- Compared with Managed Memory, CADI achieved higher performance as the problem size increased
  - CADI is effective for applications involving large-scale data transfers

Evaluation Environment

Item	Specification
CPU	AMD EPYC 7352 (2.3 GHz, 24 cores/socket, 2 sockets)
GPU	NVIDIA A100 80GB
Memory	512GB DDR4
PCI Express	Gen4 x16
OS	Rocky Linux 8.9
Compiler	NVIDIA HPC Compiler 25.3
CUDA Toolkit	CUDA Toolkit 12.8
NVIDIA Driver	version 580.126.20



Discussion

- Programming effort
  - 7 transfer targets are automatically identified for Himeno
  - Manual modifications are required for data structures that are not dynamically allocated
- Applicability
  - Structure members and function calls are supported under specific conditions
  - Indirect accesses are also supported, but the entire allocated region is transferred

Conclusion & Future Work

Conclusion

- Established a foundation for semi-automatic insertion of data transfer directives
- Automated transfer-target identification while preserving programmer control over transfer directions
- Achieved execution performance comparable to manually optimized implementations

Future Work

- Improvement of the accuracy of code analysis
- Automatic determination of transfer directions
- Performance evaluation on different hardware environments

Acknowledgement

This work was partially supported by collaborative research funding from DENSO CORPORATION. The authors also acknowledge the use of ChatGPT, developed by OpenAI to assist with proofreading and enhance manuscript clarity. The authors are accountable for the content.