IEEE CLUSTER 2025
Edinburgh, Scotland • 2–5 September

Tutorial: Runtime Performance Inefficiency Detection and Program Debugging

Yuhang Hu

Computer Network Information Center, Chinese Academy of Sciences

Hands-on Tutorial @ CLUSTER25

中国科学院计算机网络信息中心
Computer Network Information Center,
Chinese Academy of Sciences

Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
  - Installation
  - Case Study – Call Stack Obtainment
  - Case Study – Hang Detection

Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
    - Installation
    - Case Study – Call Stack Obtainment
    - Case Study – Hang Detection

Harm of Performance Variance

Applications

- computational fluid dynamics
- molecular dynamics simulations
- graph analysis
- large language models
- graph neural networks
- ……

Harm of Performance Variance

Resource Waste

Fluctuations increase average runtime by 15-40%, raising compute costs

Scientific Uncertainty

Unstable results require 3-5 times more test runs for statistical significance

Debugging Complexity

Co-existence of software and hardware errors; hard to debug due to large amount of code

System Scalability Limits

……

Challenges of Detection

Challenges of Detecting Performance Variance

Transienc

Variance occurs instantaneously and is hard to reproduce.

Dynamism

Changing patterns like sparse matrix operations.

Complexit

Multiple potential sources from hardware to software.

Vaddr

Large-scale Heterogeneous

Runtime Diagnosis

Hybrid Performance-Debugging Analysis

Irregular calculations

Low-Overhead

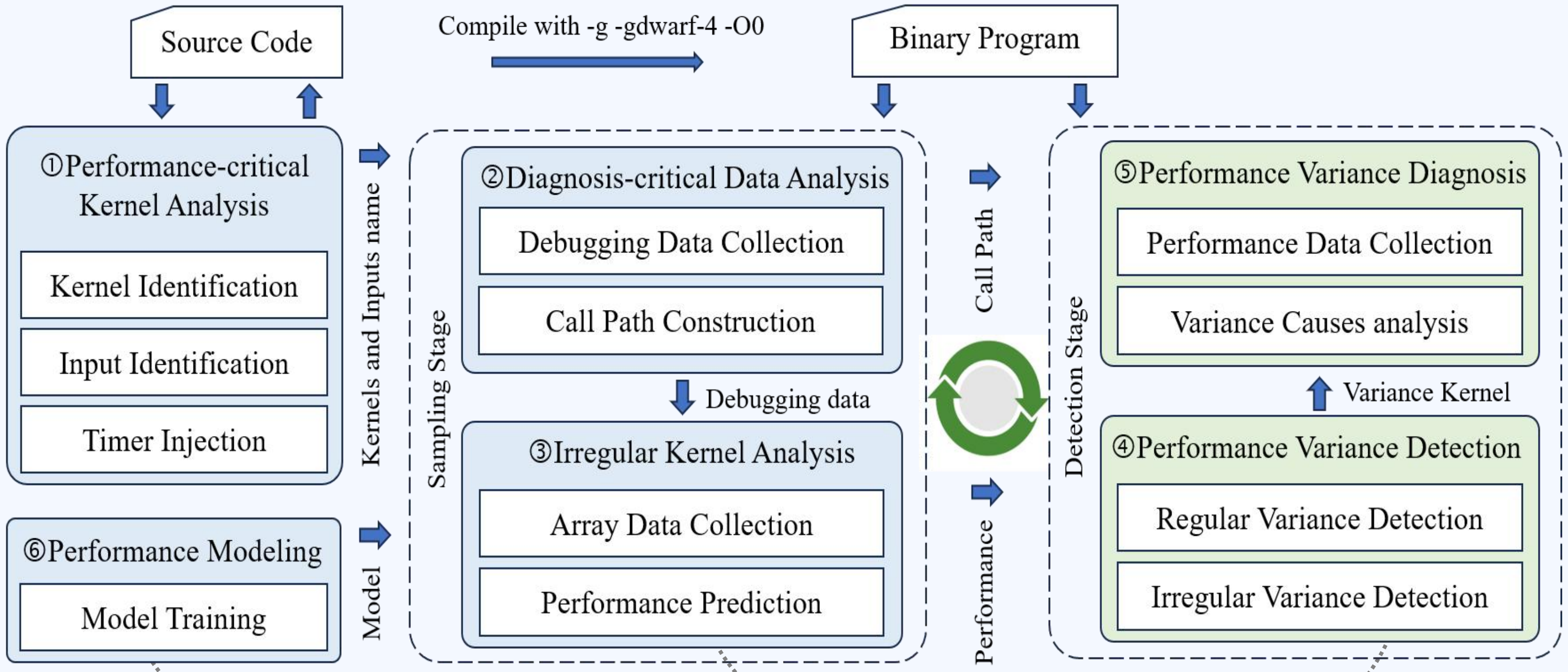
Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
  - Installation
  - Case Study – Call Stack Obtainment
  - Case Study – Hang Detection

Vaddr Overall Workflow

Vaddr: Runtime Performance Variance Detection and Diagnosis Tool

The Overview of Vaddr



Offline Analysis
Builds foundational models

Online Analysis
Performs real-time monitoring

Static Analysis Phase
- Performance-critical Kernel Analysis
- Performance Modeling

Runtime Analysis Phase
- Diagnosis-critical Data Analysis
- Irregular Kernel Analysis
- Performance Variance Detection
- Performance Variance Diagnosis

Performance-Critical Kernel Analysis

Uses **LLVM IR** to detect **CUDA/HIP** kernels

Classifies kernels as:
Regular: Predictable memory access
Irregular: Non-contiguous access patterns

01 Source to LLVM IR Translation

- DWARF debug symbols preserved **(-g -gdwarf-4)**

02 Kernel Identification

- CUDA/HIP call convention analysis
- SET_K construction
- Parameter extraction (m, n,

Regular matrix grid

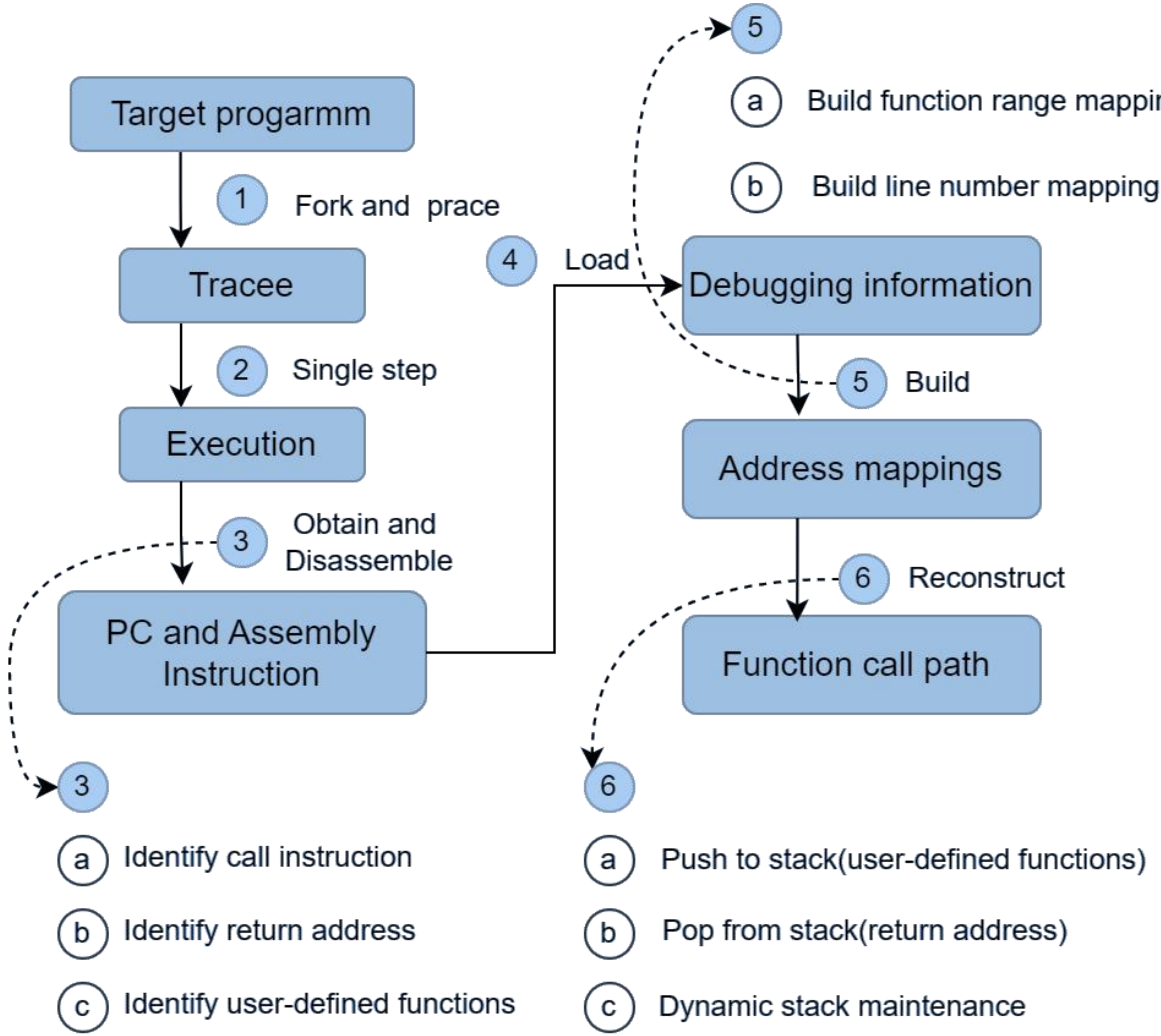
- Regular Kernel if nnz ≈ m × n
- Dense matrix operations

Irregular matrix grid

- Irregular Kernel if nnz ≪ m × n
- Sparse matrix operations

Determines Monitoring Strategy

Diagnosis-Critical Data Analysis



01 Uses Ptrace with PTRACE_SINGLESTEP for instruction-level tracing

02 Extracts DWARF debugging info (debug_line, debug_info)

03 Maps PC values to source code via

- Compilation Unit (CU) lookup
- Function boundary detection
- Line number table resolution

Irregular Kernel Analysis

Memory Introspection

DWARF-based array introspection:

- Resolves variable DIEs (Debugging Information Entries)
- Computes base addresses via CFA (Canonical Frame Address) analysis

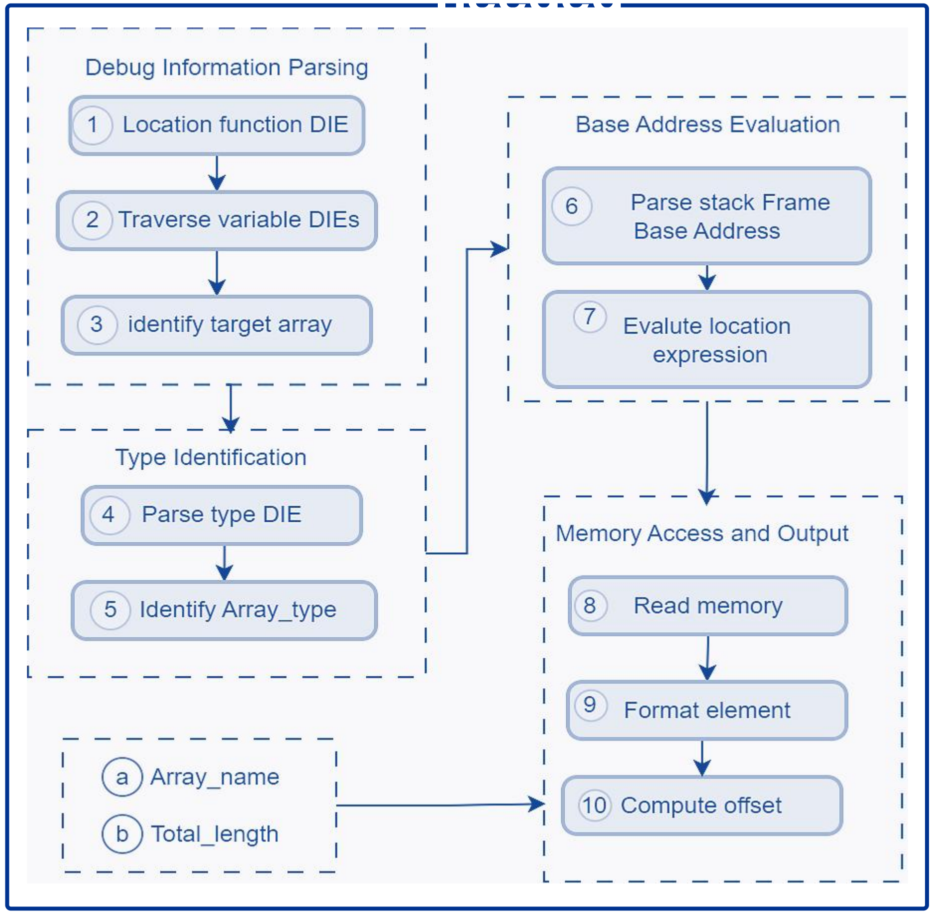
Feature Extraction

Extracts 4 feature categories:

- Matrix size (rows, cols, non-zeros)
- Nonzero skew (row/column distribution)
- Locality (Gini coefficient, average distance)
- Hardware specs (SMs, cache sizes)

Key advantage

No full matrix copy! Only address access needed



Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
  - Installation
  - Case Study – Call Stack Obtainment
  - Case Study – Hang Detection

Performance Variance Detection

1 For Regular Kernels

- Normalizes execution time by fixed-work quanta
- Identifies outliers via cross-process comparison

The **shortest duration** of same workload kernel functions as **stander time**.

2 For Irregular Kernels

- Predicts performance using ridge regression model
- Flags deviations exceeding prediction error threshold

The **predicted normal duration** of this function(related to nnz,Gini, average distance of input kernel) as **stander time.**

Threshold
Depend on experiences 10%

Performance Variance Diagnose

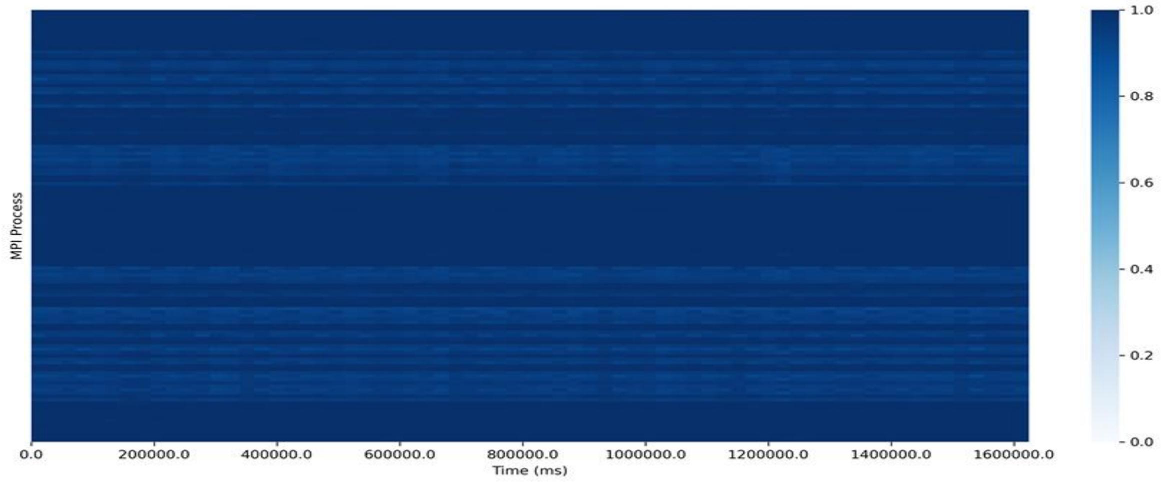
Diagnosis

1 Hardware

- Collects PMU counters via PAPI
- Normalizes metrics to [-1,1] range
- Ranks counters by Pearson correlation

2 Software

- Source code location + call path
- Adds function stack information
- Includes array contents



The light blue part shows the hardware caused performance variation.

Abnormal process details:
– File callstack–3578.txt: PC 0x400d80; Source: 39 in file /work1/wangjh/huyh/vaddr/Vaddr_case02/T_1/../target/target_modified.c

The terminal will show the software caused performance variation's rank number.

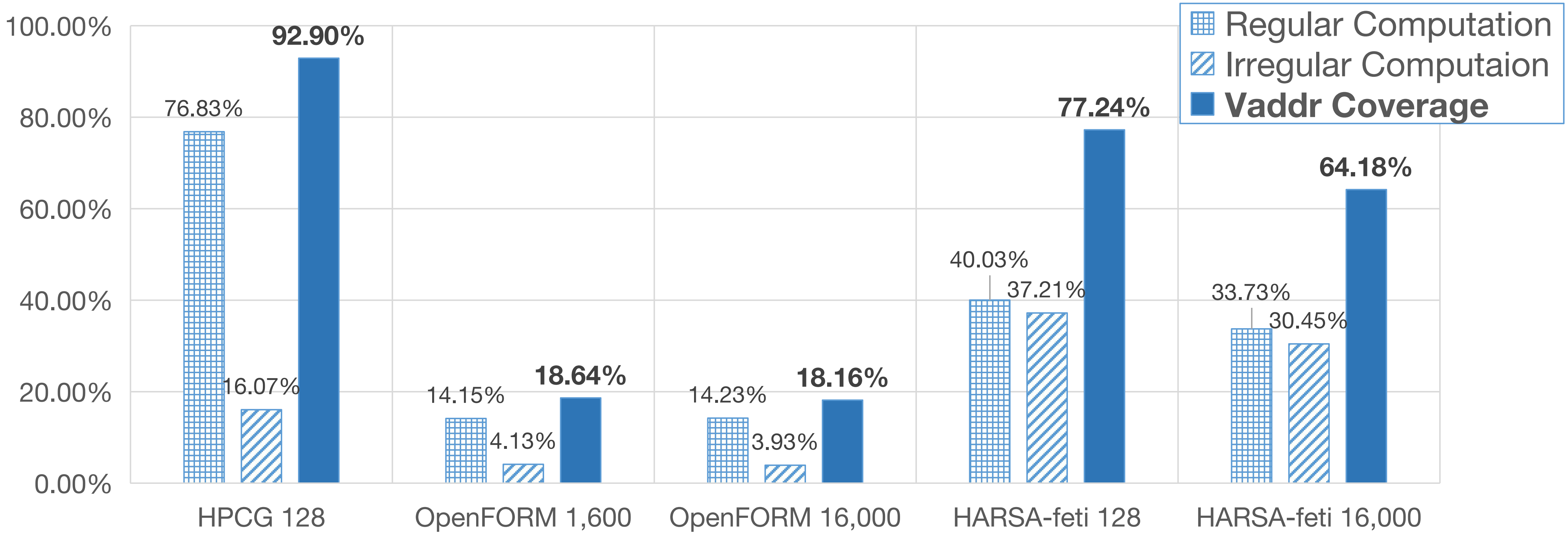
Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
  - Installation
  - Case Study – Call Stack Obtainment
  - Case Study – Hang Detection

Experimental Setup

Name	**Configuration**	
CPU	AMD Zen-based processor @ 2.5GHz, 32 cores	
GPU	4 AMD Instinct MI60 GPUs	
Memory	128 GB (CPU), 16 GB (GPU)	
Network	200 Gbps HDR InfniBand network (50Gbps x4)	
Storage	>200 Gbps	
Software	GCC 9.3.1, ROCM> 3.9, OpenMPI 4.0.4	
Evaluation Program	HPCG	A high-performance conjugate gradients benchmark
	OpenFORM	A widely adopted computational fluid dynamics software
	HARSA-feti	A scalable structural dynamics simulator in various domains

Coverage



- Since computation workload on GPU in HPCG,OpenFORM and HARSA-feti include sparse matrix computation, the performance variance detection method for regular computation workload, only detect regular computation workload.
- Vaddr can identify irregular computation workload, thereby achieving a certain level of detection coverage.

Overhead

The detection and diagnosis overhead of Vaddr

Program	Scale	Monitor	Sampling	Analysis	Time	Storage
HPCG	128	20.10%	23.06%	56.84%	1.12x(179.04 s)	0.65GB
OpenFORM	16,000	21.47%	22.35%	56.18%	1.16x(903.08 s)	4.06GB
HARSA-feti	16,000	22.05%	21.33%	56.62%	1.14x(789.12 s)	3.58GB

The overhead of Vaddr consists of three parts, including monitor overhead, storage overhead, and analysis overhead.

- **Monitor overhead:** Refers to the ratio of the execution time with Vaddr of the execution time without Vaddr.
- **Storage overhead:** Refers to the storage of the program tracing data, including performance and debugging data.
- **Analysis overhead:** Refers to the absolute time in seconds required for Vaddr analysis.

Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
    - Installation
    - Case Study – Call Stack Obtainment
    - Case Study – Hang Detection

Installation - Vaddr

- **Install in tutorial cluster:**
  - Dependencies： gcc-7.3.1, gcc-11.2.1, g++, make, cmake-3.24.3, zycore-c-1.5.1, zydis, papi-7.0.0, hip-5.2, clang-14.0.0, llvm-14.0.0, libunwind, libunwind, libelfin, linenoise, C++17
  - Source Code Package :

    ```
    cp /work1/wangjh/huyh/vaddr/Vaddr.tar.gz ./
    ```

  - Compilation Instruction：

    ```
    tar -xzvf Vaddr.tar.gz
    ```

    ```
    cd Vaddr
    ```

- **Instruction to analyze the target program ：**

```
./run.sh
```

Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
    - Installation
    - Case Study – Call Stack Obtainment
    - Case Study – Hang Detection

Case Study – Call Stack Obtainment (~5min)

- Get the Vaddr:

```
cp /work1/wangjh/huyh/vaddr/Vaddr_case01.tar.gz ./
```

```
tar -xzvf Vaddr_case01.tar.gz
```

```
cd Vaddr_case01
```

- Run for the first time

```
./run.sh
```

- Save the call stac

```
cp -r ./T_234/build/output/ ./output1
```

- Modify the target program

```
vim target/target.cpp
```

```cpp
int main(int argc, char** argv) {
    int rank, size;

    // f();
    // int foo=7;

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Uncomment

```cpp
int main(int argc, char** argv) {
    int rank, size;

    f();
    int foo=7;

    // Initialize MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
```

Case Study – Call Stack Obtainment

- Modify the target program

Press "I" to "--INSERT--" mode

Uncomment line 50 & 51

Press "ESC" to exit "--INSERT--" mode

Press ":wq" to save the modification and exit

- Run for the second time

```
./run.sh
```

- Save the call stack

```
cp -r ./T_234/build/output/ ./output2
```

```
output1  output2  run.sh  T_1  T_234  target
```

Call stack for the first time

Call stack for the second time

Case Study – Call Stack Obtainment

- Resulting files are generated in **./T_234/build/output/** folder
- Files are named with **callstack-<PID>.txt**

Time of the operation

The line number of the current operation

Current PC value

Path to the code file

Function name of the operation

```
[2025–08–05 10:51:11.074] Debugger started
run util main PC: 0x400d79
arrive at main entry point
Linenormal: 19 in file /work1/wangjh/wangky/ptrace/T_1/../target/target_modified.cpp
function call: main

[2025–08–05 10:51:11.117] step line
Current PC: 0x400d78
Linenormal: 19 in file /work1/wangjh/wangky/ptrace/T_1/../target/target_modified.cpp
function call: main
Received SIGTRAP at 0x400d7a

[2025–08–05 10:51:11.117] step line
Current PC: 0x400d7b
Linenormal: 19 in file /work1/wangjh/wangky/ptrace/T_1/../target/target_modified.cpp
function call: main
Received SIGTRAP at 0x400d7f
```

callstack-<pid>.txt

Case Study – Call Stack Obtainment

- The call stack of **target.cpp with nested function**.

target/target_modified.cpp

```
9  void a() {
10     int foo = 1;
11 }
12
13 void b() {
14     int foo = 2;
15     a();
16 }
17
18 void c() {
19     int foo = 3;
20      b();
21 }
22
……
```

You will see the process of **entering and exiting nested functions**, and the information provided **corresponds to the source code line numbers.**

```
...
Received SIGTRAP at 0x400f72
 current_pc1: 0x400f73
  depth: 6 -- custom func in line: 11 in file
/work1/wangjh/huyh/vaddr/Vaddr_case01/T_1/../target/target_modified.cpp
function call: main->f()->e()->d()->c()->b()->a()
Received SIGTRAP at 0x400fca
 current_pc1: 0x400fcb
  depth: 6 -- custom func in line: 16 in file
/work1/wangjh/huyh/vaddr/Vaddr_case01/T_1/../target/target_modified.cpp
function call: main->f()->e()->d()->c()->b()
Received SIGTRAP at 0x400f9a
 current_pc1: 0x400f9b
  depth: 5 -- custom func in line: 21 in file
/work1/wangjh/huyh/vaddr/Vaddr_case01/T_1/../target/target_modified.cpp
function call: main->f()->e()->d()->c()
Received SIGTRAP at 0x400fff
 current_pc1: 0x401000
  depth: 3 -- custom func in line: 26 in file
/work1/wangjh/huyh/vaddr/Vaddr_case01/T_1/../target/target_modified.cpp
function call: main->f()->e()->d()
...
```

output2/callstack-<pid>.txt

Outline

- Introduction & Background
- Understanding of Vaddr Workflow
- Performance Variance Detection
- Evaluation
- Hands-on Tutorial
  - Installation
  - Case Study – Call Stack Obtainment
  - Case Study – Hang Detection

Case Study – Hang Detection(~ 10min)

- Get the Vaddr:

```
/work1/wangjh/huyh/vaddr/Vaddr_case02.tar.gz ./
```

```
tar -xzvf Vaddr_case02.tar.gz
```

```
cd Vaddr_case02
```

- Run Vaddr:

```
./run.sh
```

target/target_modified.cpp

```
36   MPI_Bcast(h_in, N * N, MPI_FLOAT, 0,
MPI_COMM_WORLD);
37
38     if (rank == 1) {
39         sleep(10);
40     }
41
42     hipMemcpy(d_in, h_in, N * N * sizeof(float),
hipMemcpyHostToDevice);
```

Corresponding line number

Terminal Output

```
Abnormal process details:
– File callstack–3578.txt: PC 0x400d80;
Source: 39 in file
/work1/wangjh/huyh/vaddr/Vaddr_case02/T_1/../target/target_modified.c
Saved CSV to pauses.csv
```

Case Study – Hang Detection(~ 10min)

- Modify the target program

vim target/target.cpp

Press "I" to "--INSERT--" mode

Comment / uncomment code

Press "ESC" to exit "--INSERT--" mode

Press ":wq" to save the modification and exit

For ease of observation, UNCOMMENT only ONE sleep() at a time.

target/target_modified.cpp

```
if (rank == 0) {
//  sleep(10);
    std::cout << "Output Matrix:" << std::endl;
     for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
             std::cout << h_out[i * N + j] << " ";
        }
        std::cout << std::endl;
    }
}
// if (rank == 3) {
//     sleep(10);
// }
```

Uncomment

```
if (rank == 0) {
//  sleep(10);
    std::cout << "Output Matrix:" << std::endl;
     for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
             std::cout << h_out[i * N + j] << " ";
        }
        std::cout << std::endl;
    }
}
if (rank == 3) {
   sleep(10);
}
```

Case Study – Hang Detection

- Run Vaddr: `./run.sh`
- Result show in the terminal.

Terminal Output

```
Abnormal process details:
– File callstack–3578.txt: PC 0x400d80; Source: 39 in file
/work1/wangjh/huyh/vaddr/Vaddr_case02/T_1/../target/target_modified.c
Saved CSV to pauses.csv
```

Line number of the hang appeared

The path of the code

Terminal Output

```
No long pauses detected (min=9.0s, max=11.5s). Program appears normal.
Saved CSV to pauses.csv
```

The program is running normally

Thanks!

Yuhang Hu

Computer Network Information Center, Chinese Academy of Sciences