# oclude and OCLMan

## tools to profile and predict the dynamic behavior of standalone OpenCL kernels based on compiling and machine learning techniques

Sotirios Niarchos

School of Electrical and Computer Engineering
National Technical University of Athens
Division of Computer Science

Computer Systems Laboratory (CSLab)

July 29, 2020

# Outline I

1 Introduction
- It is a heterogeneous world
- Utilizing diversity
- Related work

2 oclude
- The need for a profiler
- A glimpse of OpenCL
- An overview of oclude

# Outline II

3 OCLBoi
- Towards the instcounts model
- The design of OCLBoi
- OCLBoi and the Rodinia Suite

4 OCLMan
- A boy needs a father
- The design of OCLMan
- Evaluating OCLMan

5 Future work

# What is heterogeneous computing?

# What is heterogeneous computing?

## Towards a definition (1/2)

*"Todays computing environments are becoming more multifaceted, exploiting the capabilities of a range of* **multi-core microprocessors**, *central processing units (***CPUs***), digital signal processors, reconfigurable hardware (***FPGAs***), and graphic processing units (***GPUs***)."*[1]

[1] Gaster, Benedict et al. *Heterogeneous Computing with OpenCL*. 1st ed. 2011.

# What is heterogeneous computing?

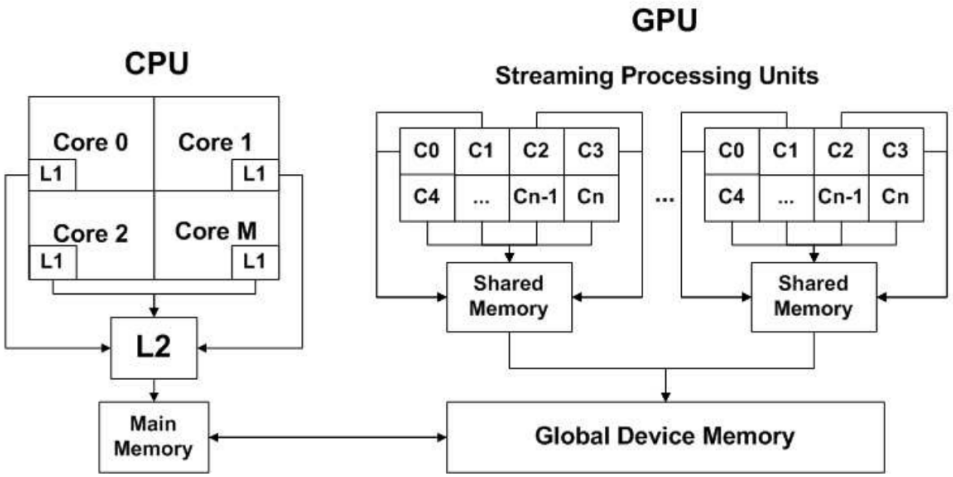



Figure: A simple heterogeneous system[1]

[1] Ravi, Vignesh et al. *Proceedings of the International Conference on Supercomputing*. "Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations". 2010.

# What is heterogeneous computing?

## Towards a definition (2/2)

*"The definition of this term is quite straightforward:*
**executing programs on a computing platform with computing nodes of different characteristics**.

*What is tricky is whether this is a good thing or a bad thing."*[1]

[1] Zahran, Mohamed. *Heterogeneous Computing: Hardware & Software Perspectives*. 2019.

# What is heterogeneous computing?

## Towards a definition (2/2)

*"The definition of this term is quite straightforward:*
**executing programs on a computing platform with computing nodes of different characteristics**.

*What is tricky is whether this is a good thing or a bad thing."*[1]

[1] Zahran, Mohamed. *Heterogeneous Computing: Hardware & Software Perspectives*. 2019.

# Valuable potential...

*"Heterogeneous computer systems [. . . ] add richness by allowing the programmer to select the best architecture to execute the task at hand or to choose the right task to make optimal use of a given architecture"*[2]

[2] Gaster, Benedict et al. *Heterogeneous Computing with OpenCL*. 1st ed. 2011.

# Valuable potential...

*"Heterogeneous computer systems [. . . ] add richness by allowing the programmer to select the best architecture to execute the task at hand or to choose the right task to make optimal use of a given architecture"*[2]

[2] Gaster, Benedict et al. *Heterogeneous Computing with OpenCL*. 1st ed. 2011.

# Valuable potential...

*"Heterogeneous computer systems [. . . ] add richness by allowing the programmer to select the best architecture to execute the task at hand or to choose the right task to make optimal use of a given architecture"*[2]

[2] Gaster, Benedict et al. *Heterogeneous Computing with OpenCL*. 1st ed. 2011.

# ...if we learn how to use it

- **How** to select the best architecture for a given task?

# ...if we learn how to use it

- **How** to select the best architecture for a given task?
- **How** to select the right task for a given architecture?

# ...if we learn how to use it

- **How** to select the best architecture for a given task?
- **How** to select the right task for a given architecture?

Non-trivial tasks...

# ...if we learn how to use it

- **How** to select the best architecture for a given task?
- **How** to select the right task for a given architecture?

Non-trivial tasks... unless we manage to predict the **execution time** of a *specific* application on a *specific* processing unit

# ...if we learn how to use it

- Related literature **agrees on the necessity** of execution time prediction...

# ...if we learn how to use it

- Related literature **agrees on the necessity** of execution time prediction...
- ...but has not agreed on **how** to do it.

# ...if we learn how to use it

- Related literature **agrees on the necessity** of execution time prediction...
- ...but has not agreed on **how** to do it.
- Our work is **a novel approach** on this subject

# ...if we learn how to use it

- Related literature **agrees on the necessity** of execution time prediction...
- ...but has not agreed on **how** to do it.
- Our work is **a novel approach** on this subject
- We will be working with the OpenCL framework for heterogeneous computation...

# ...if we learn how to use it

- Related literature **agrees on the necessity** of execution time prediction...
- ...but has not agreed on **how** to do it.
- Our work is **a novel approach** on this subject
- We will be working with the OpenCL framework for heterogeneous computation...
- ... but we will **not be limited by it!**

# The dominant approach

*What to use and how to use it in order to predict execution time?*

# The dominant approach

*What to use and how to use it in order to predict execution time?*

- **static** source code features (e.g. # of instructions, # of basic blocks etc.)[a]

[a] Wen, Yuan, Wang, Zheng, and O'Boyle, Michael. "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms". 2014.

[b] Heckmann, Reinhold and Ferdinand, Christian. *International Federation for Information Processing Digital Library; Building the Information Society;* "aiT: Worst-Case Execution Time Prediction by Static Program Analysis". 2004.

# The dominant approach

*What to use and how to use it in order to predict execution time?*

- **static** source code features (e.g. # of instructions, # of basic blocks etc.)[a]
- heavy **source code analysis** (e.g. loop bound analysis, path analysis etc.)[b]

---

[a] Wen, Yuan, Wang, Zheng, and O'Boyle, Michael. "Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms". 2014.

[b] Heckmann, Reinhold and Ferdinand, Christian. *International Federation for Information Processing Digital Library; Building the Information Society;* "aiT: Worst-Case Execution Time Prediction by Static Program Analysis". 2004.

# The dominant approach

*What to use and how to use it in order to predict execution time?*

- However, building analytical models has been deemed obsolete[a], due to:
  1. the **complexity** of the process
  2. **over-simplistic assumptions** that are needed

[a] Huang, Ling et al. "Predicting Execution Time of Computer Programs Using Sparse Polynomial Regression". 2010.

# An alternative approach

*What to use and how to use it in order to predict execution time?*

# An alternative approach

*What to use and how to use it in order to predict execution time?*

- **dynamic/runtime** program features (e.g. # of **executed** instructions)

# An alternative approach

*What to use and how to use it in order to predict execution time?*

- **dynamic/runtime** program features (e.g. # of **executed** instructions)
  - **implicitly combine** static features and source code analysis

# An alternative approach

*What to use and how to use it in order to predict execution time?*

- **dynamic/runtime** program features (e.g. # of **executed** instructions)
  - **implicitly combine** static features and source code analysis
  - uncover the **runtime behavior** of the application

# An alternative approach

*How to extract dynamic features from an application?*

[3] Yang, L. T., Xiaosong Ma, and Mueller, F. "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution". 2005.

[4] Chun, Byung-Gon et al. "Mantis: Predicting System Performance through Program Analysis and Modeling". 2010.

# An alternative approach

*How to extract dynamic features from an application?*

- **partial execution**[3]: *"very short testdrives of applications on multiple candidate platforms to quickly derive the execution time of much longer runs."*

[3] Yang, L. T., Xiaosong Ma, and Mueller, F. "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution". 2005.

[4] Chun, Byung-Gon et al. "Mantis: Predicting System Performance through Program Analysis and Modeling". 2010.

# An alternative approach

*How to extract dynamic features from an application?*

- **partial execution**[3]: *"very short testdrives of applications on multiple candidate platforms to quickly derive the execution time of much longer runs."*
- **instrumentation** and **feature evaluators**[4]: *"automatically extract small code snippets (feature evaluators) that compute feature values from the instrumented program."*

[3] Yang, L. T., Xiaosong Ma, and Mueller, F. "Cross-Platform Performance Prediction of Parallel Applications Using Partial Execution". 2005.

[4] Chun, Byung-Gon et al. "Mantis: Predicting System Performance through Program Analysis and Modeling". 2010.

# From input size to execution time

$$dynamic\ \ features \longmapsto t_{exec}$$

# From input size to execution time

$$input\ size \longmapsto dynamic\ features \longmapsto t_{exec}$$

# From input size to execution time

$$input\ size \longmapsto \textbf{\textit{instcounts}} \longmapsto t_{exec}$$

# From input size to execution time

$$\boldsymbol{gsize} \longmapsto \boldsymbol{instcounts} \longmapsto t_{exec}$$

# Decoupling input size and execution time

- $gsize \longmapsto instcounts$ : application-specific, hardware-agnostic
- $instcounts \longmapsto t_{exec}$ : application-agnostic, hardware-specific

# Decoupling input size and execution time

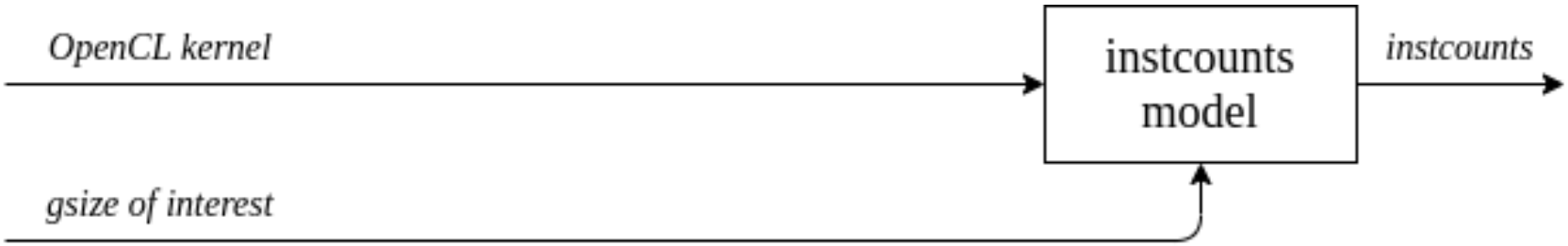
## Main goal

*Predict instcounts from gsize for a given OpenCL kernel*

# Decoupling input size and execution time

## Main goal

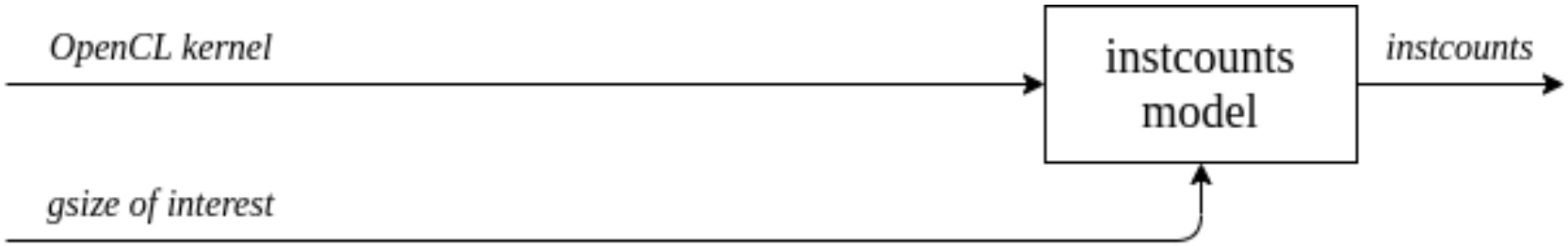
*Predict instcounts from gsize for a given OpenCL kernel*



# Decoupling input size and execution time

## Main goal

*Predict instcounts from gsize for a given OpenCL kernel*




- *Something* is needed to extract dynamic information from the OpenCL kernel in order to train **OCLBoi**, the instcounts model...

# Decoupling input size and execution time

## Main goal

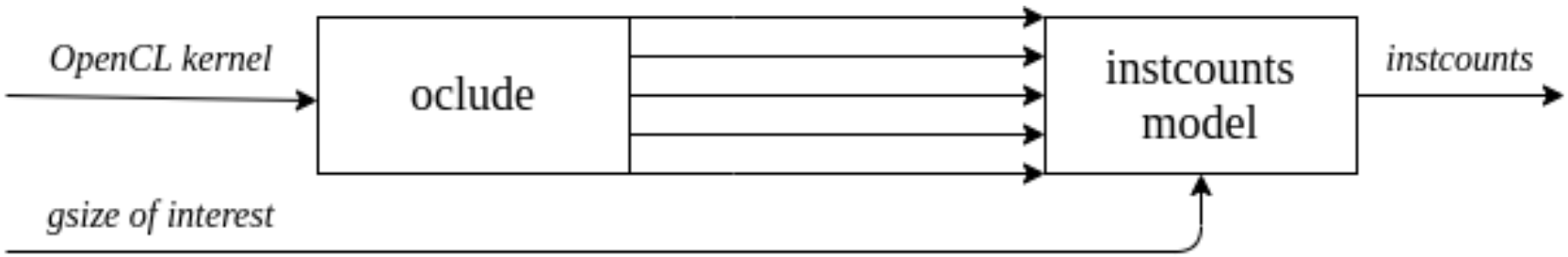
*Predict instcounts from gsize for a given OpenCL kernel*


- *Something* is needed to extract dynamic information from the OpenCL kernel in order to train **OCLBoi**, the instcounts model...

# Decoupling input size and execution time

## Main goal

*Predict instcounts from gsize for a given OpenCL kernel*

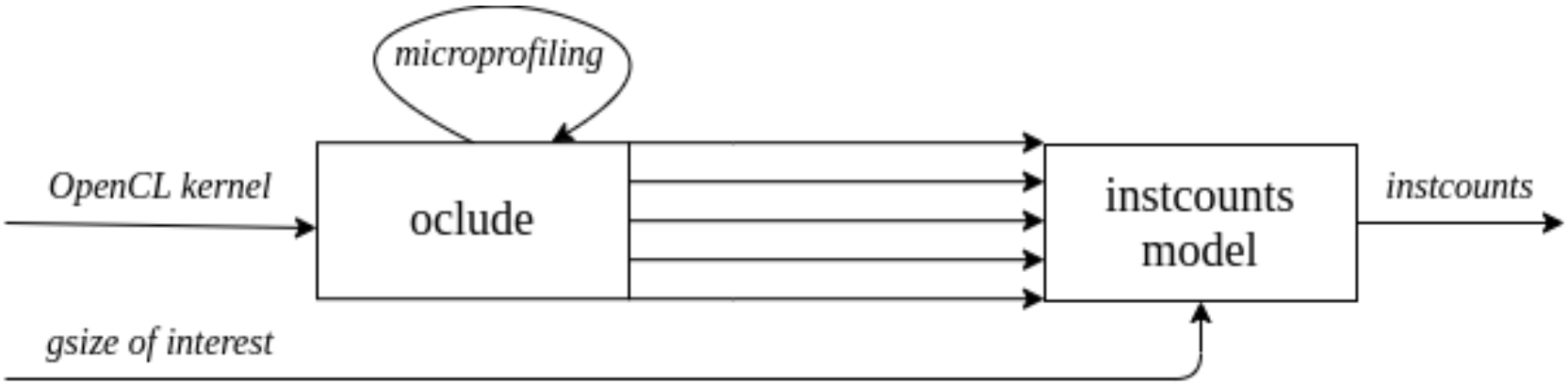



- *Something* is needed to extract dynamic information from the OpenCL kernel in order to train **OCLBoi**, the instcounts model...
- ...and that something is **oclude**.

# Decoupling input size and execution time

## Main goal

*Predict instcounts from gsize for a given OpenCL kernel*

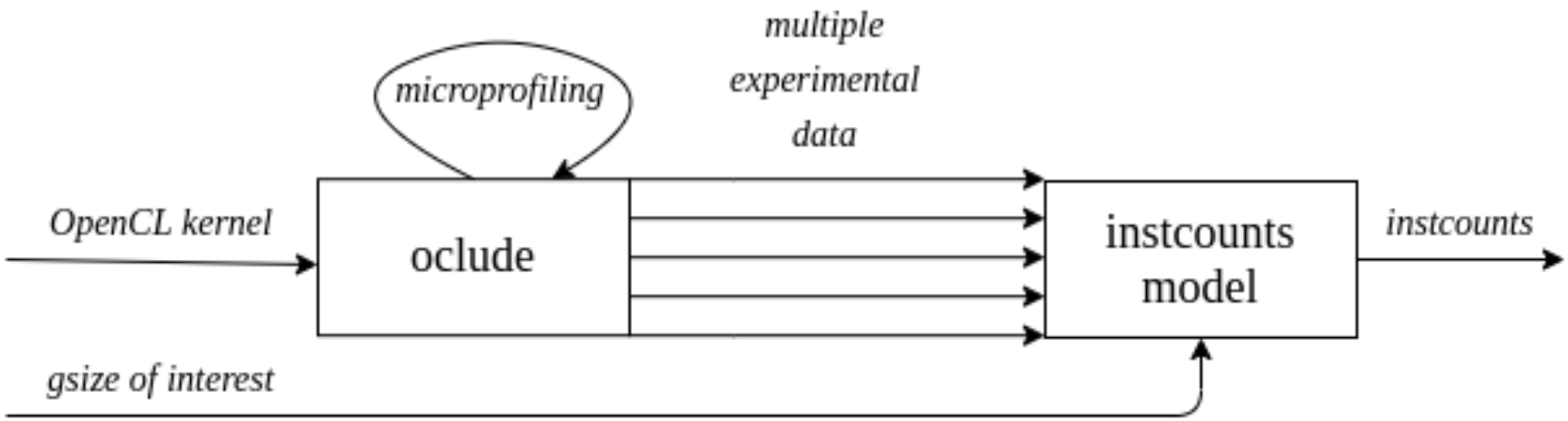



- *Something* is needed to extract dynamic information from the OpenCL kernel in order to train **OCLBoi**, the instcounts model...
- ...and that something is **oclude**.

# Decoupling input size and execution time

## Main goal

*Predict instcounts from gsize for a given OpenCL kernel*




- *Something* is needed to extract dynamic information from the OpenCL kernel in order to train **OCLBoi**, the instcounts model...

# What is OpenCL?


- OpenCL is a **specification** for heterogeneous computation by Khronos Group Inc.

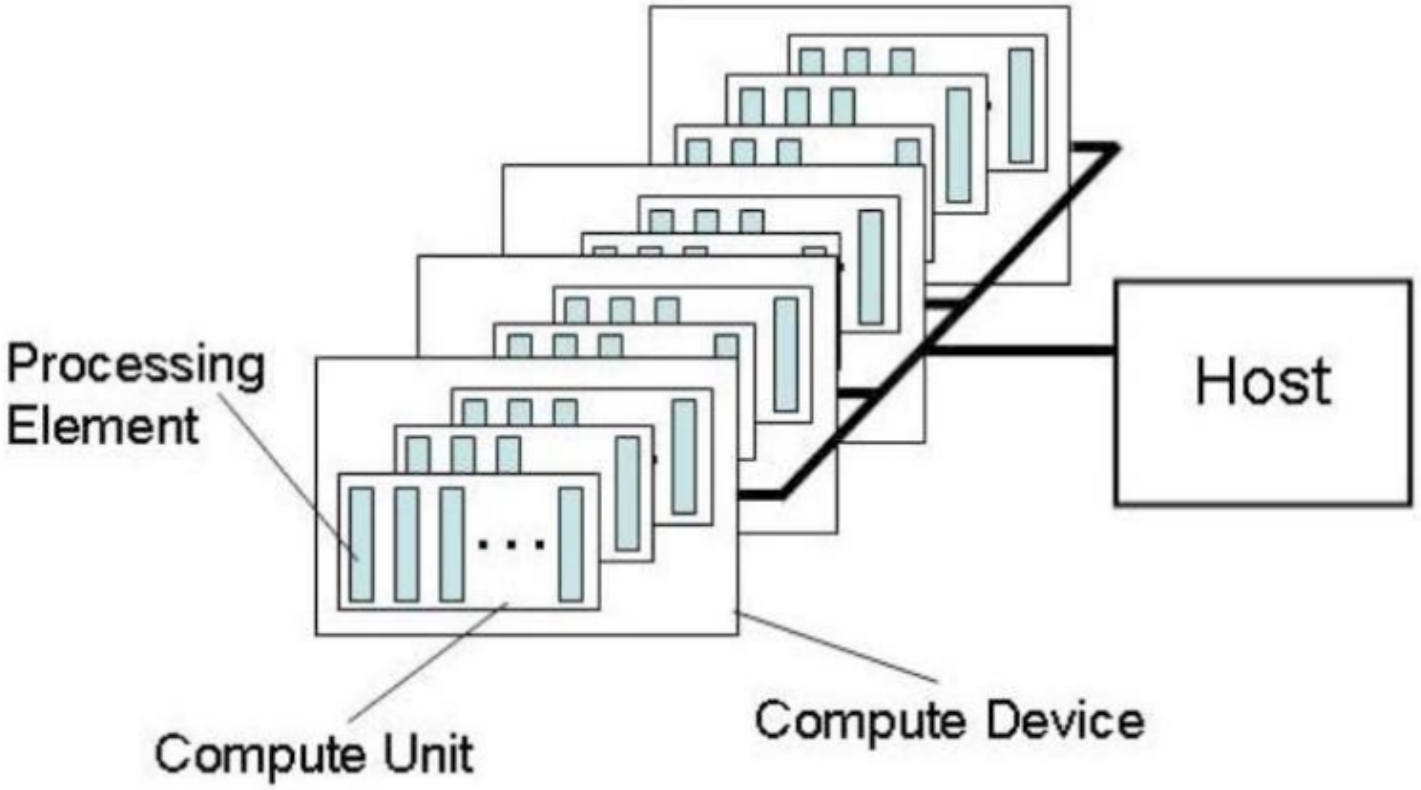
# What is OpenCL?


- OpenCL is a **specification** for heterogeneous computation by Khronos Group Inc.
- OpenCL proposes:
  1. *(to the users)* a way to design, create and run **applications** on parallel/heterogeneous systems
  2. *(to hardware vendors)* protocols that **processing units** (CPUs, GPUs, etc) must follow in order to facilitate the above

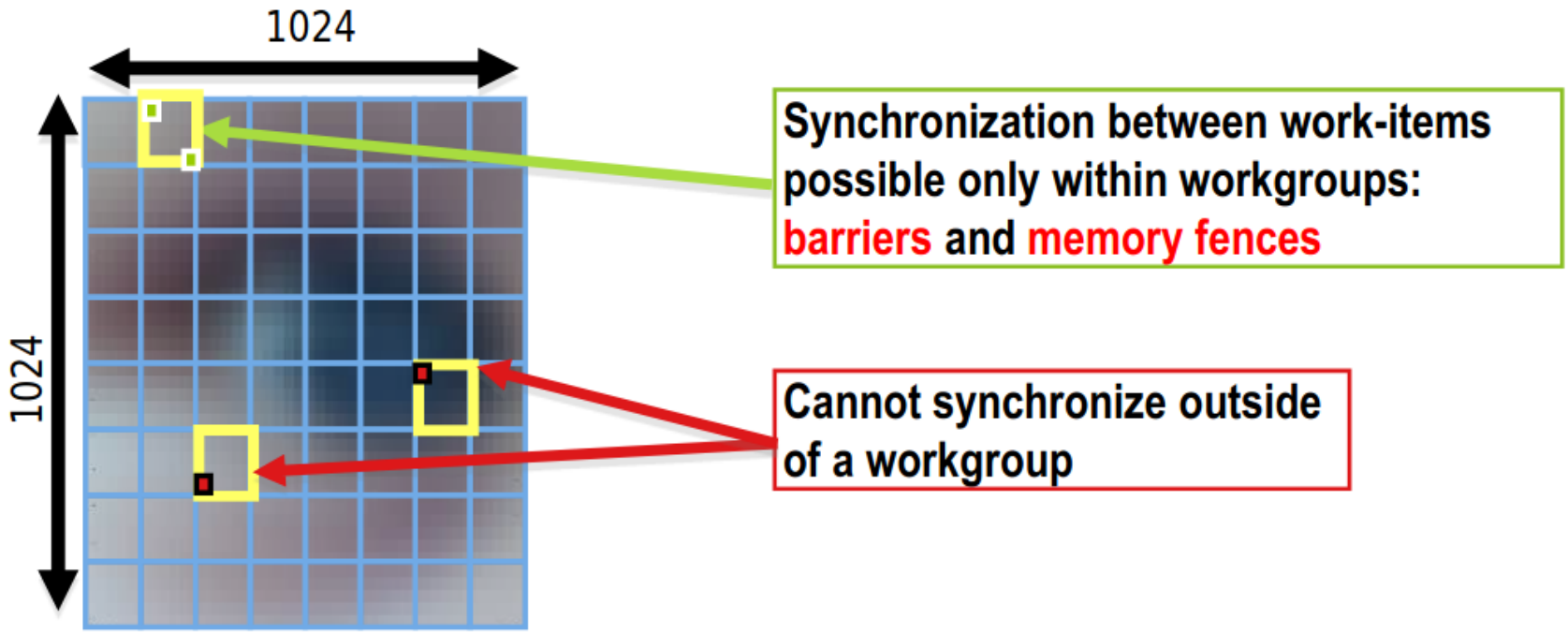
# What is OpenCL?


- OpenCL is a **specification** for heterogeneous computation by Khronos Group Inc.
- OpenCL proposes:
  1. *(to the users)* a way to design, create and run **applications** on parallel/heterogeneous systems
  2. *(to hardware vendors)* protocols that **processing units** (CPUs, GPUs, etc) must follow in order to facilitate the above
- it is **not** a specific implementation

# The OpenCL execution model

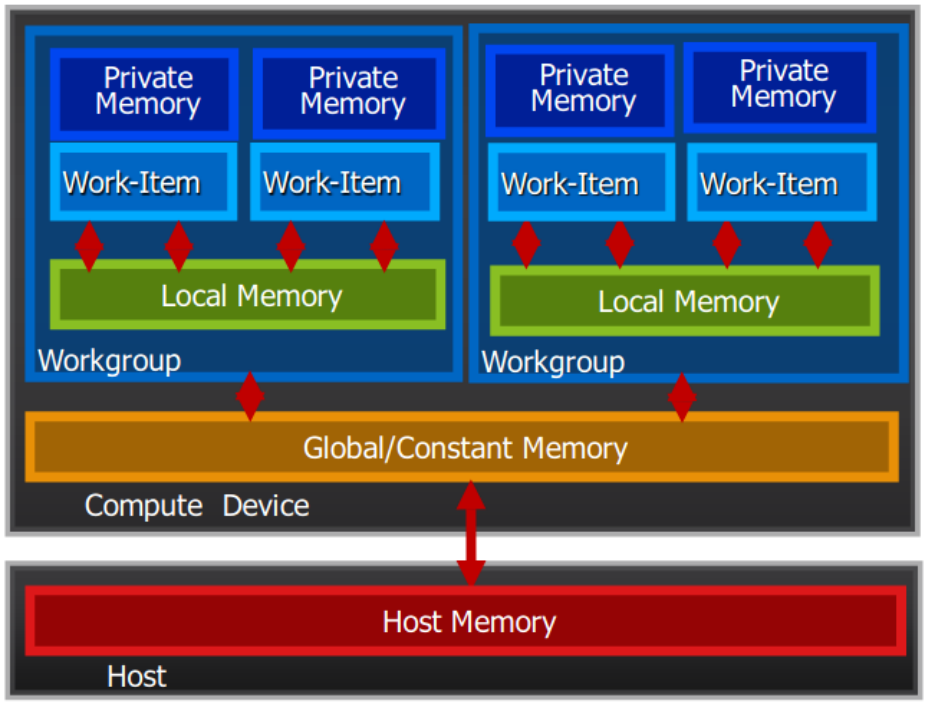


# The OpenCL task grid

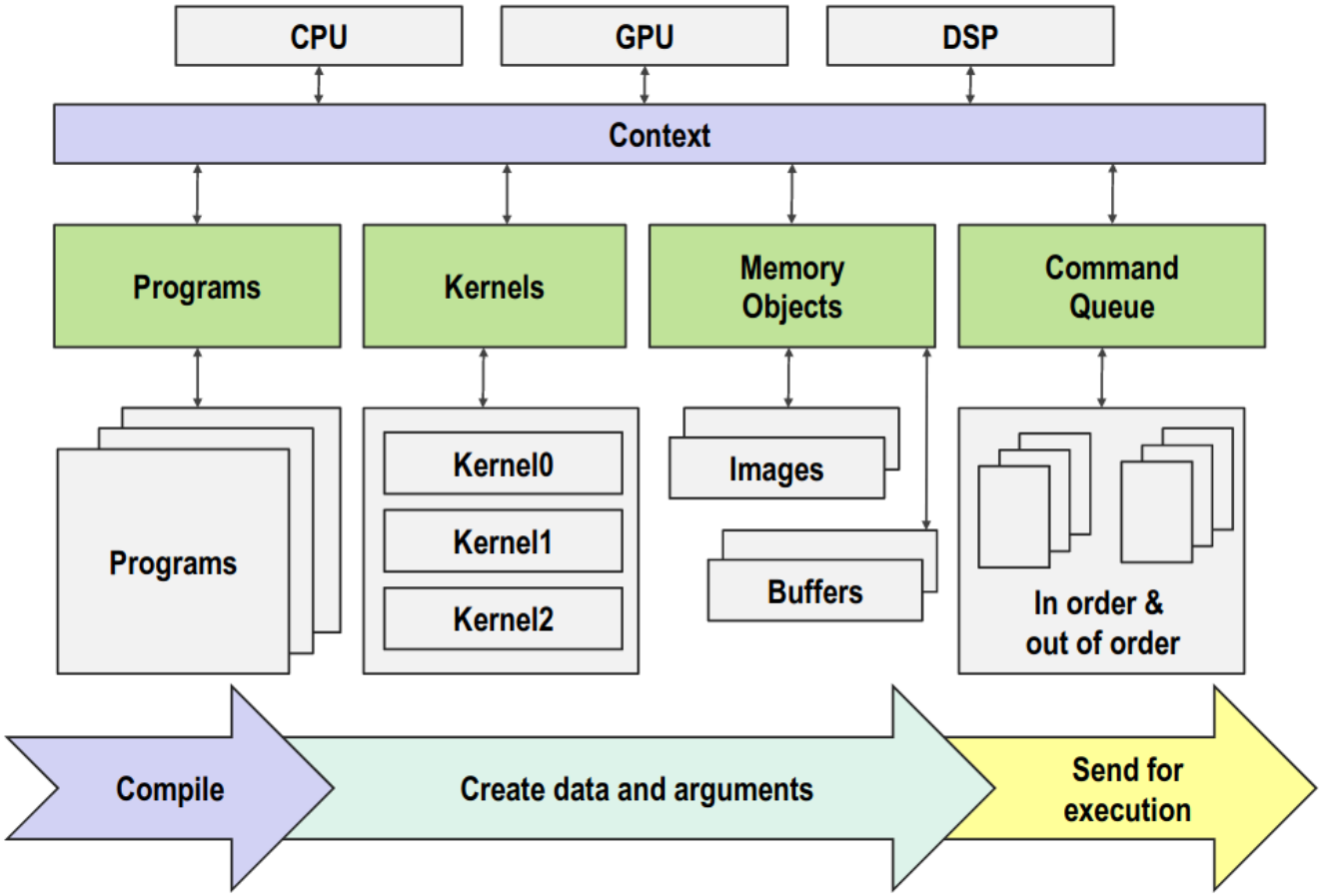


# The OpenCL memory model

- **Private Memory**
  - per *work-item*
- **Local Memory**
  - shared within a *workgroup*
- **Global/Constant Memory**
  - visible to all workgroups
- **Host Memory**
  - on the CPU



# A complete overview of the OpenCL workflow



# An overview of oclude

**What it is**

- An open-source standalone OpenCL kernel runner and profiler[5]
- The most technically challenging component of our work
- `Python 3`, `C++`
- Ways to use it:
  1. As a **command line utility** on Unix-like OSs
  2. As a **Python package**

[5] `https://github.com/zehanort/oclude`

# An overview of oclude

## What it does

- In our work, **dynamic features = executed LLVM instructions (instcounts)**

[5] `https://github.com/zehanort/rvg`

[6] Klöckner, Andreas et al. *Parallel Computing*. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". 2012.

# An overview of oclude

## What it does

- In our work, **dynamic features = executed LLVM instructions (instcounts)**
- oclude workflow

[5] `https://github.com/zehanort/rvg`

[6] Klöckner, Andreas et al. *Parallel Computing*. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". 2012.

# An overview of oclude

## What it does

- In our work, **dynamic features = executed LLVM instructions (instcounts)**
- oclude workflow
  1. compilation to **LLVM bitcode** and extraction of (static) instruction counts

[5] `https://github.com/zehanort/rvg`

[6] Klöckner, Andreas et al. *Parallel Computing*. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". 2012.

# An overview of oclude

## What it does

- In our work, **dynamic features = executed LLVM instructions (instcounts)**
- oclude workflow
  1. compilation to **LLVM bitcode** and extraction of (static) instruction counts
  2. **source code instrumentation** *(make the kernel count the instructions it executes)*

[5] `https://github.com/zehanort/rvg`

[6] Klöckner, Andreas et al. *Parallel Computing*. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". 2012.

# An overview of oclude

## What it does

- In our work, **dynamic features = executed LLVM instructions (instcounts)**
- oclude workflow
  1. compilation to **LLVM bitcode** and extraction of (static) instruction counts
  2. **source code instrumentation** *(make the kernel count the instructions it executes)*
  3. random argument initialization[5] based on **gsize**

[5] `https://github.com/zehanort/rvg`

[6] Klöckner, Andreas et al. *Parallel Computing*. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". 2012.

# An overview of oclude

## What it does

- In our work, **dynamic features = executed LLVM instructions (instcounts)**
- oclude workflow
  1. compilation to **LLVM bitcode** and extraction of (static) instruction counts
  2. **source code instrumentation** *(make the kernel count the instructions it executes)*
  3. random argument initialization[5] based on **gsize**
  4. kernel execution through the **PyOpenCL API**[6]

[5] `https://github.com/zehanort/rvg`

[6] Klöckner, Andreas et al. *Parallel Computing*. "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation". 2012.

# An overview of oclude

## An example of usage

```
$ oclude -f com_dwt.cl -k c_CopySrcToComponents -g 1024 -it
... (info on standard error) ...
Instructions executed for kernel 'c_CopySrcToComponents':
           20480 - load private
           14336 - alloca
           14336 - store private
           12288 - add
           11264 - mul
            9216 - getelementptr
            9216 - sext
            4096 - call
            3072 - load global
            3072 - load local
            3072 - store local
            3072 - zext
            2048 - trunc
            1024 - ret
            1024 - br
            1024 - icmp
Time measurement info regarding the execution for kernel 'c_CopySrcToComponents' (in milliseconds):
hostcode - 7.42030143737793
  device - 5.3919999999999995
transfer - 2.0283014373779302
```

# An overview of oclude

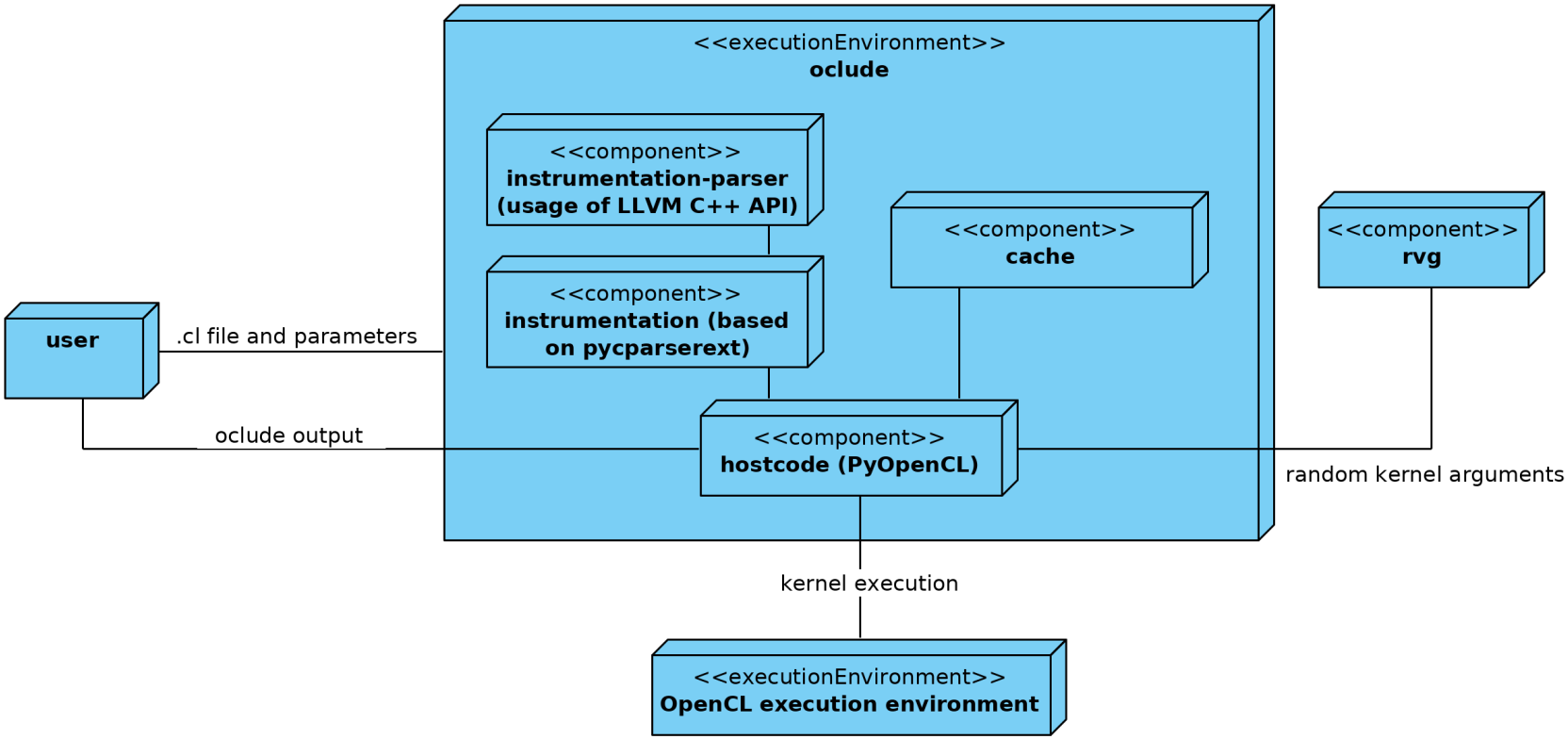


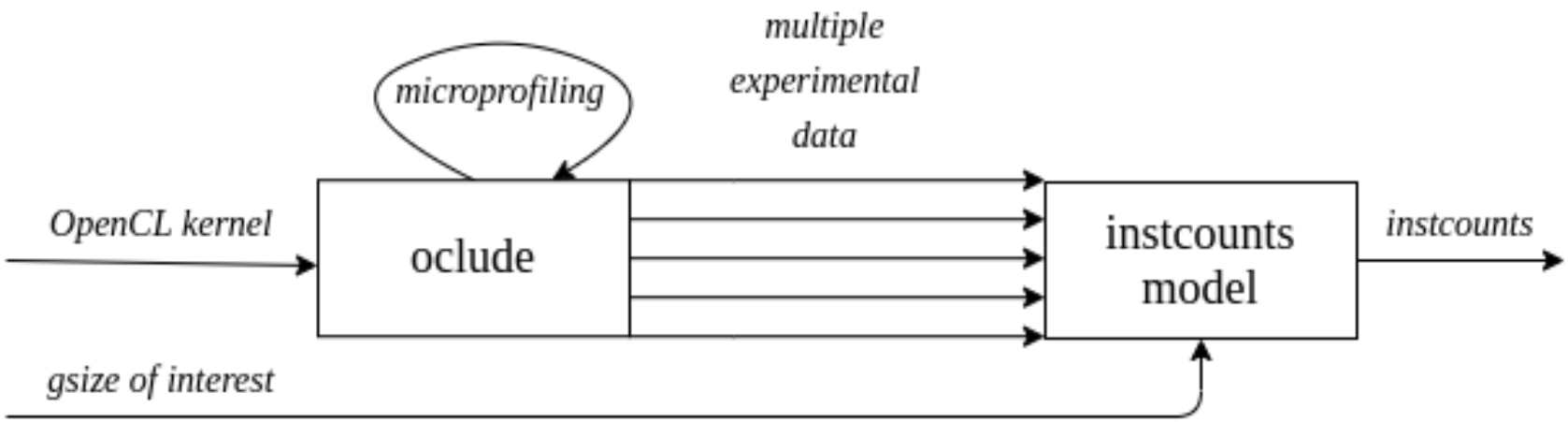

Figure: oclude UML component diagram

# Before and after instrumentation

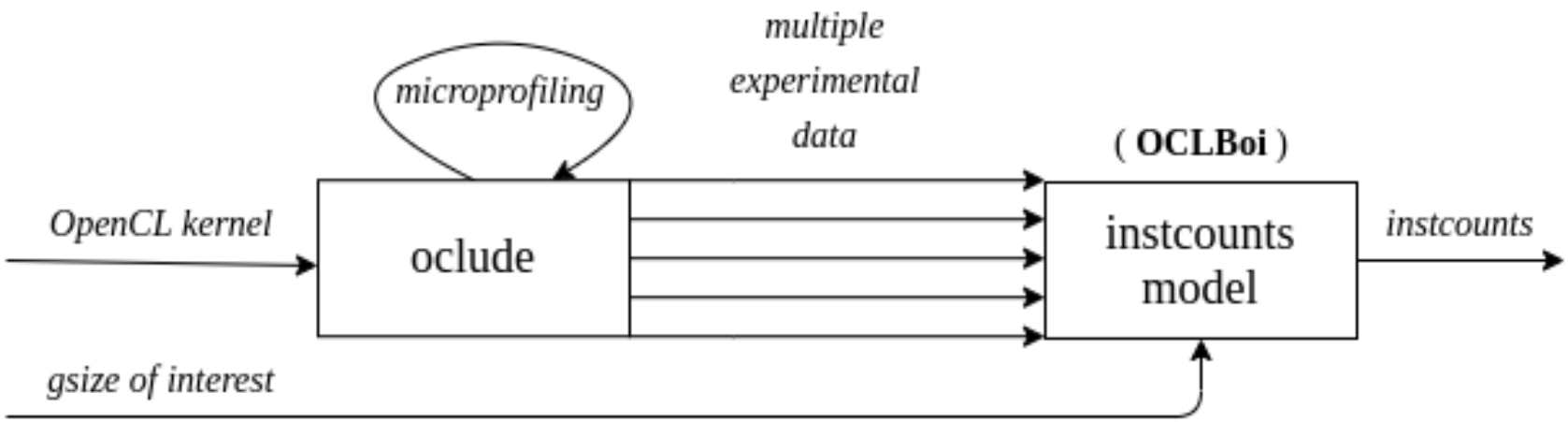
```
__kernel void
vadd(__global int *a,
     __global int *b,
     __global int *c) {

int i = get_global_id(0);
c[i] = a[i] +  b[i];
}
```

```
__kernel void
vadd(__global int *a,
     __global int *b,
     __global int *c,
     __local ulong *ocludeHiddenCounterLocal,
     __global ulong *ocludeHiddenCounterGlobal) {

  if (get_local_id(0) == 0)
    for (int i = 0; i < 73; i++)
    ocludeHiddenCounterLocal[i] = 0;
  barrier(CLK_GLOBAL_MEM_FENCE);

  /* alloca */
  atom_add(& ocludeHiddenCounterLocal[24], 6);
  /* store private */
  atom_add(& ocludeHiddenCounterLocal[30], 6);
  ...
  int i = get_global_id(0);
  c[i] = a[i] + b[i];

  barrier(CLK_GLOBAL_MEM_FENCE);
  if (get_local_id(0) == 0)
    for (int i = 0; i < 73; i++)
    atom_add(& ocludeHiddenCounterGlobal[i],
             ocludeHiddenCounterLocal[i]);
}
```

# A quick reminder



# A quick reminder



# Profiling kernels with oclude

The experimental process

- We worked with the OpenCL kernels of the **Rodinia Benchmark Suite**[7]

[7] Che, S. et al. "Rodinia: A benchmark suite for heterogeneous computing". 2009.

# Profiling kernels with oclude

The experimental process

- We worked with the OpenCL kernels of the **Rodinia Benchmark Suite**[7]
- We profiled each kernel for **a range of gsizes**

[7] Che, S. et al. "Rodinia: A benchmark suite for heterogeneous computing". 2009.

# Profiling kernels with oclude

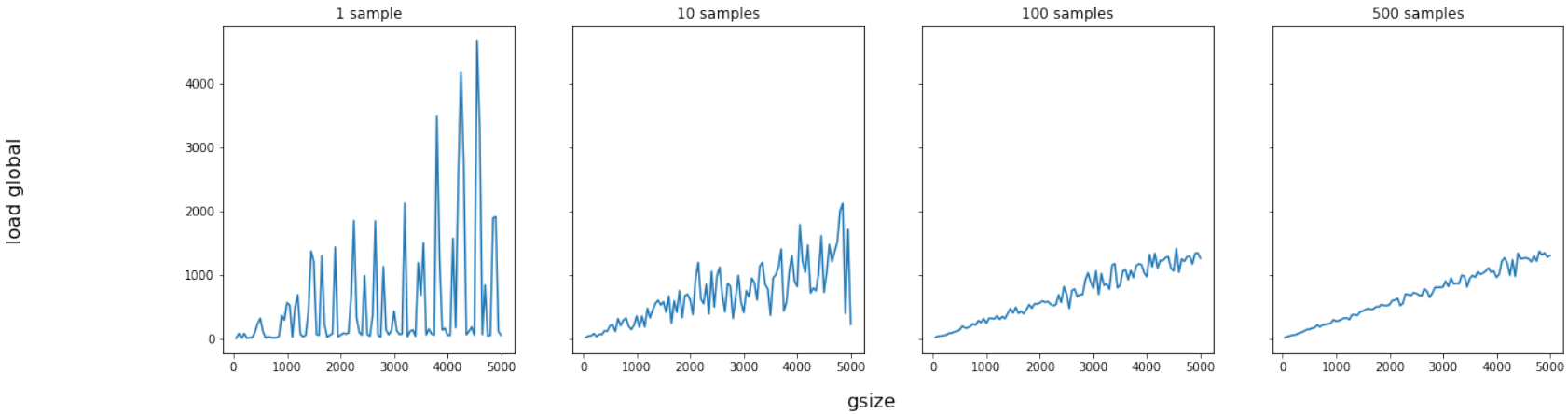
The experimental process

- We worked with the OpenCL kernels of the **Rodinia Benchmark Suite**[7]
- We profiled each kernel for **a range of gsizes**
- We took **100 samples for each gsize value**

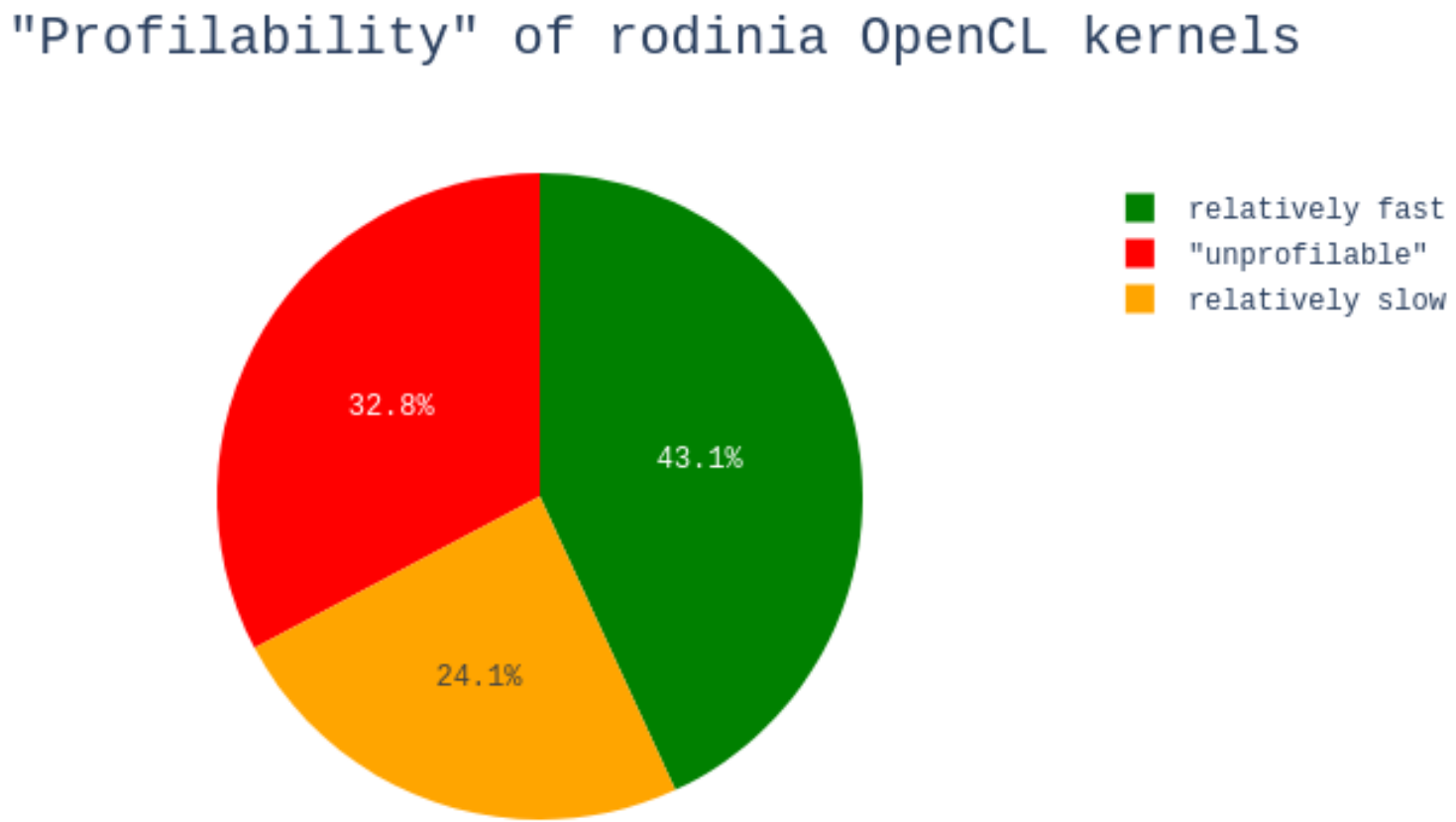
[7] Che, S. et al. "Rodinia: A benchmark suite for heterogeneous computing". 2009.

# Profiling kernels with oclude

- Why 100 samples?

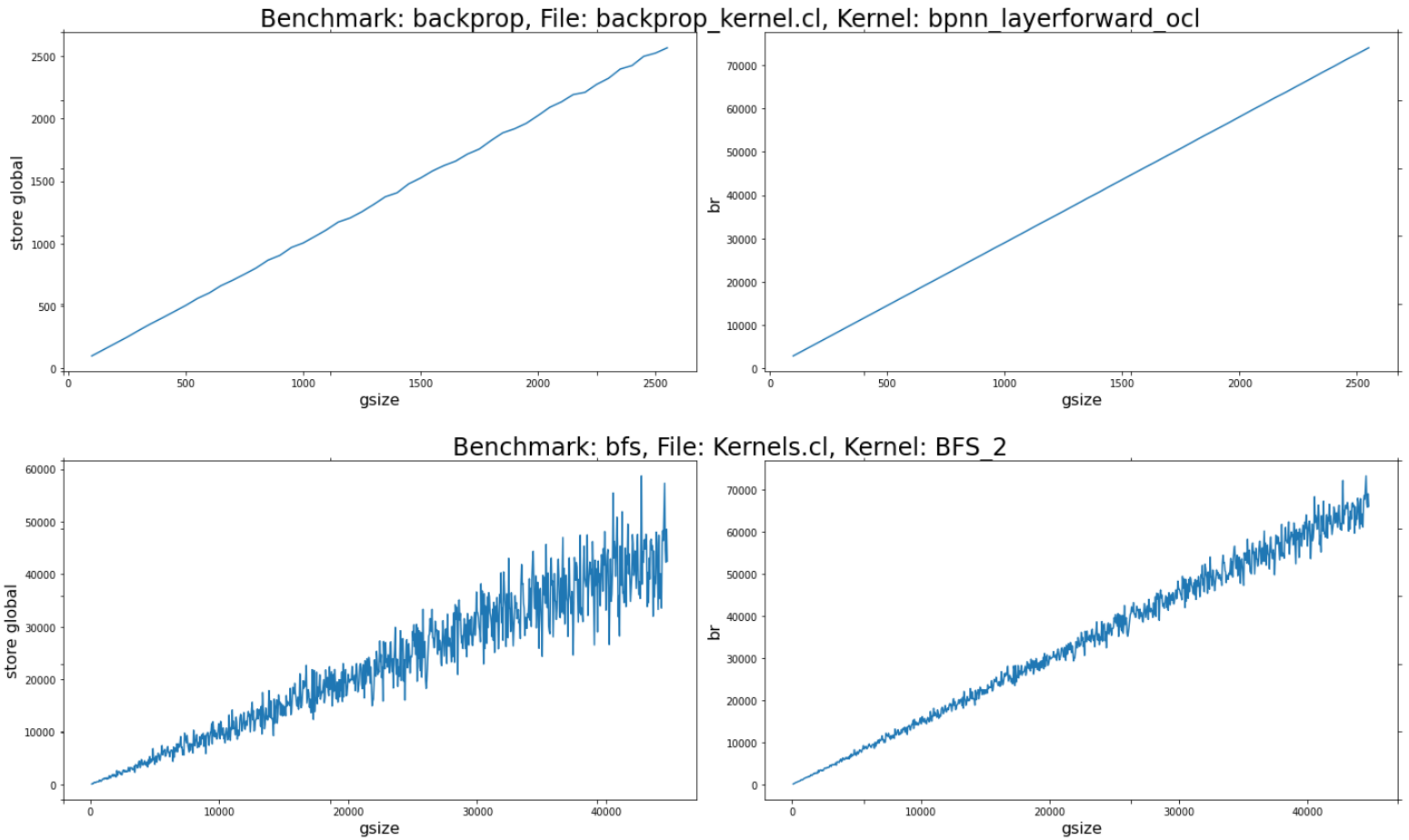


# Exploratory data analysis on Rodinia measurements



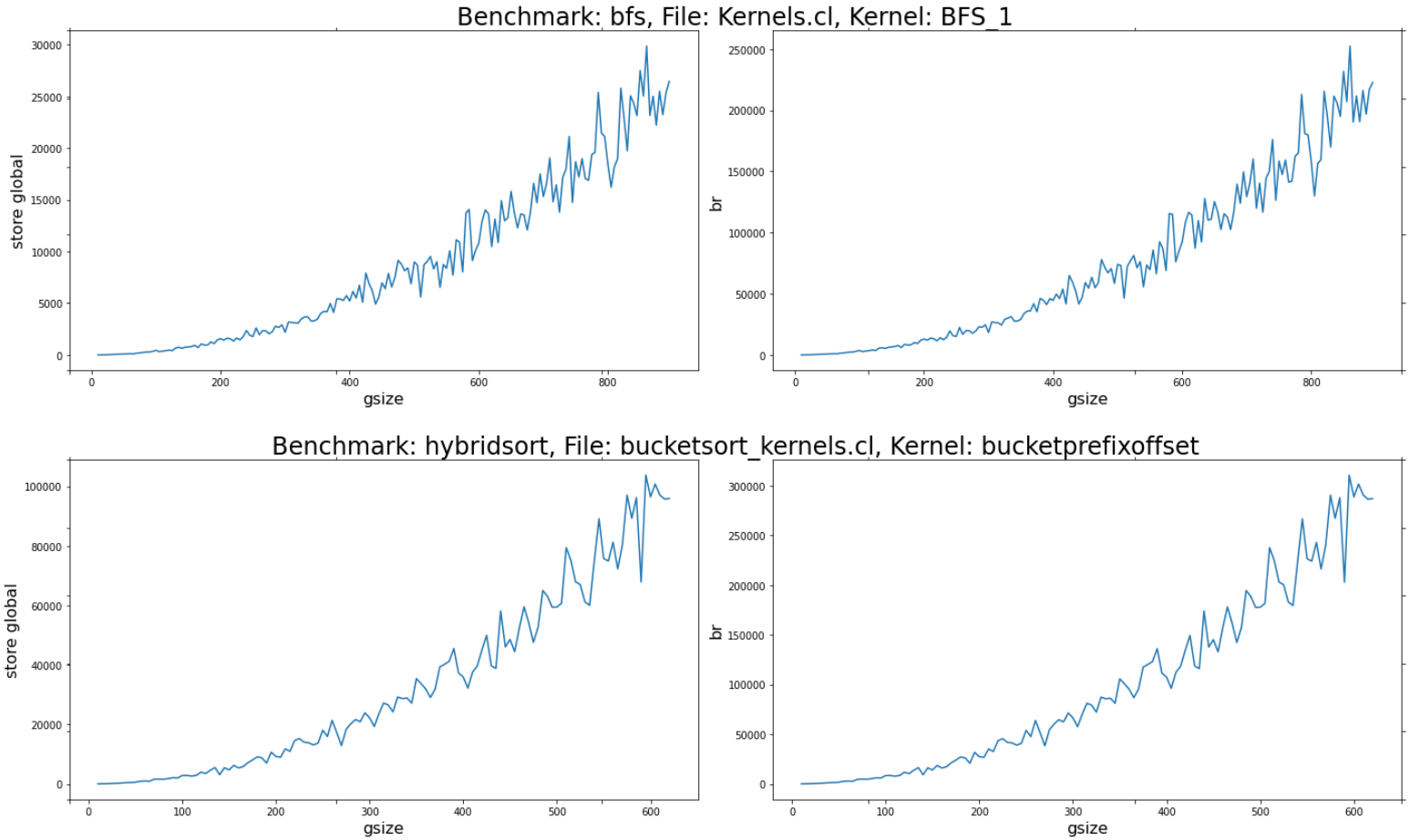
# Exploratory data analysis on Rodinia measurements

## Some "relatively fast" kernels

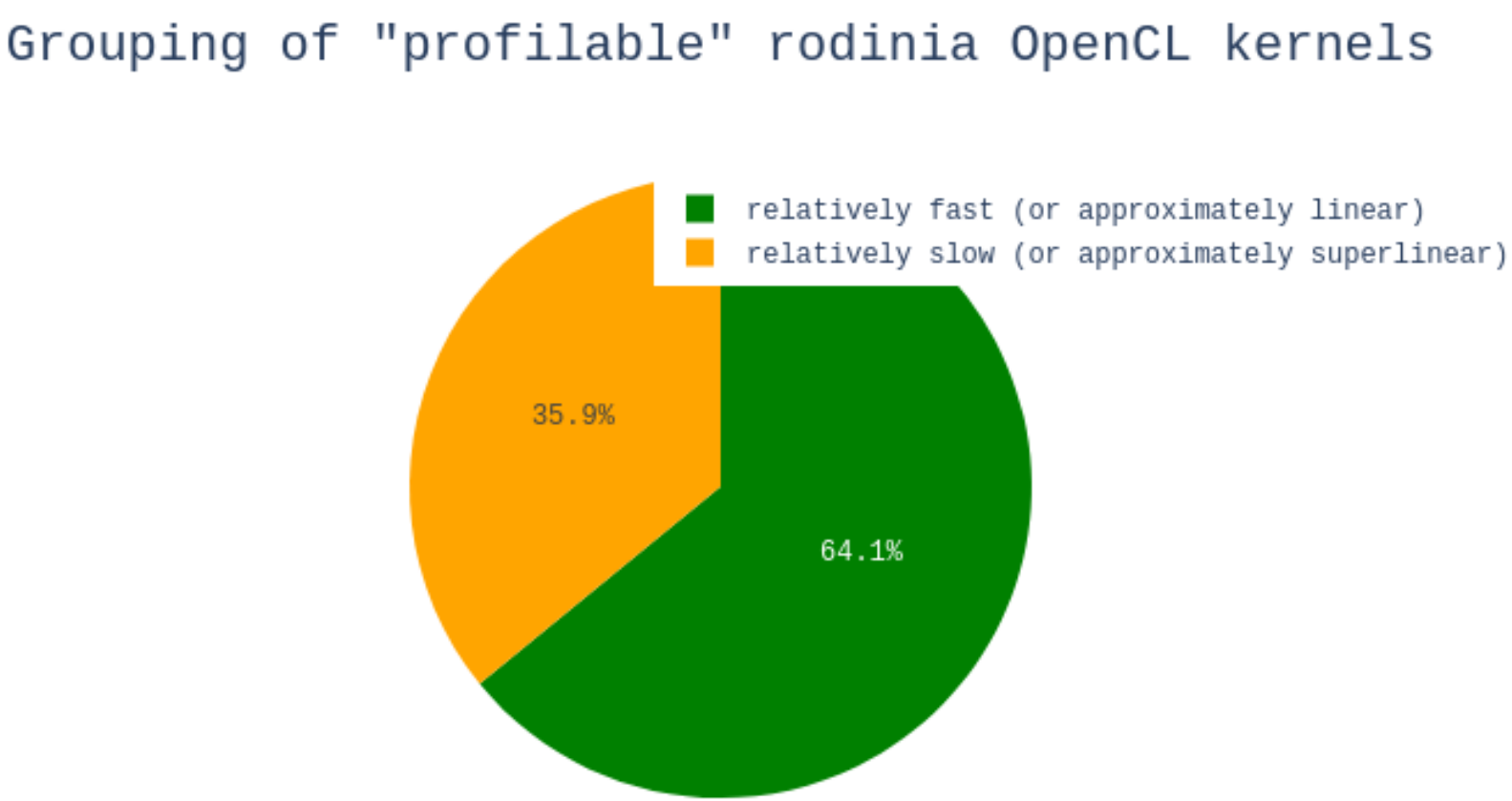


# Exploratory data analysis on Rodinia measurements

## Some "relatively slow" kernels



# Exploratory data analysis on Rodinia measurements



# Exploratory data analysis on Rodinia measurements

Therefore, can we estimate the nature of the relationship between **gsize** and **instcounts**?

# Exploratory data analysis on Rodinia measurements

Therefore, can we estimate the nature of the relationship between **gsize** and **instcounts**?

- "relatively fast" $\rightarrow$ **linear relationship**
- "relatively slow" $\rightarrow$ **polynomial relationship up to degree 2**

# The design of OCLBoi

- **OCLBoi** (*"OpenCL, But One In-particular"*) is our instcounts model

# The design of OCLBoi

- **OCLBoi** (*"OpenCL, But One In-particular"*) is our instcounts model
- **kernel-specific** *(one in particular!)*, **hardware-agnostic**

# The design of OCLBoi

- **OCLBoi** (*"OpenCL, But One In-particular"*) is our instcounts model
- **kernel-specific** *(one in particular!)*, **hardware-agnostic**
- predicts **instcounts** based on a **gsize** value

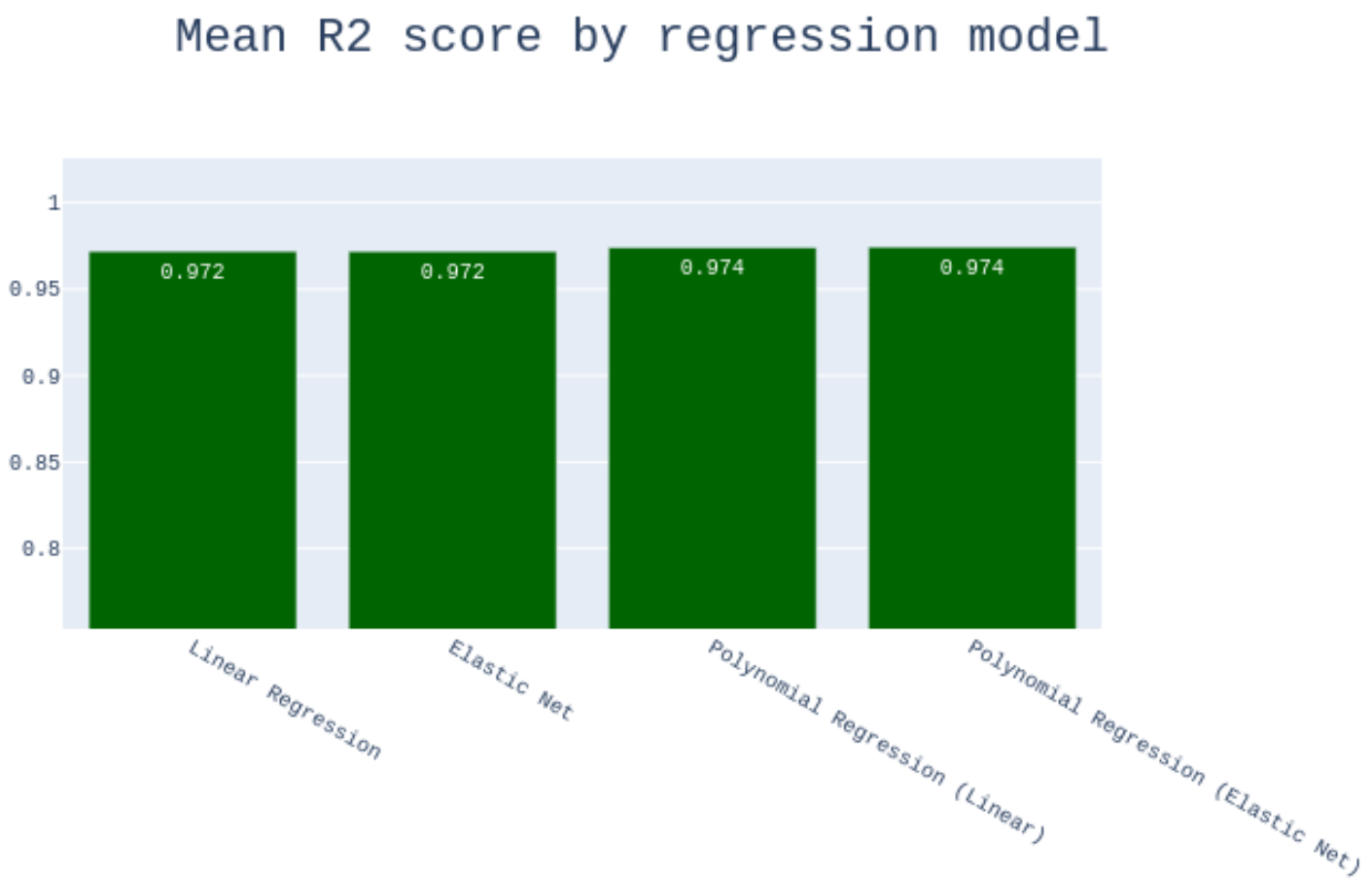
# The design of OCLBoi

- **OCLBoi** (*"OpenCL, But One In-particular"*) is our instcounts model
- **kernel-specific** *(one in particular!)*, **hardware-agnostic**
- predicts **instcounts** based on a **gsize** value
- training and testing on the measurements extracted from Rodinia via oclude

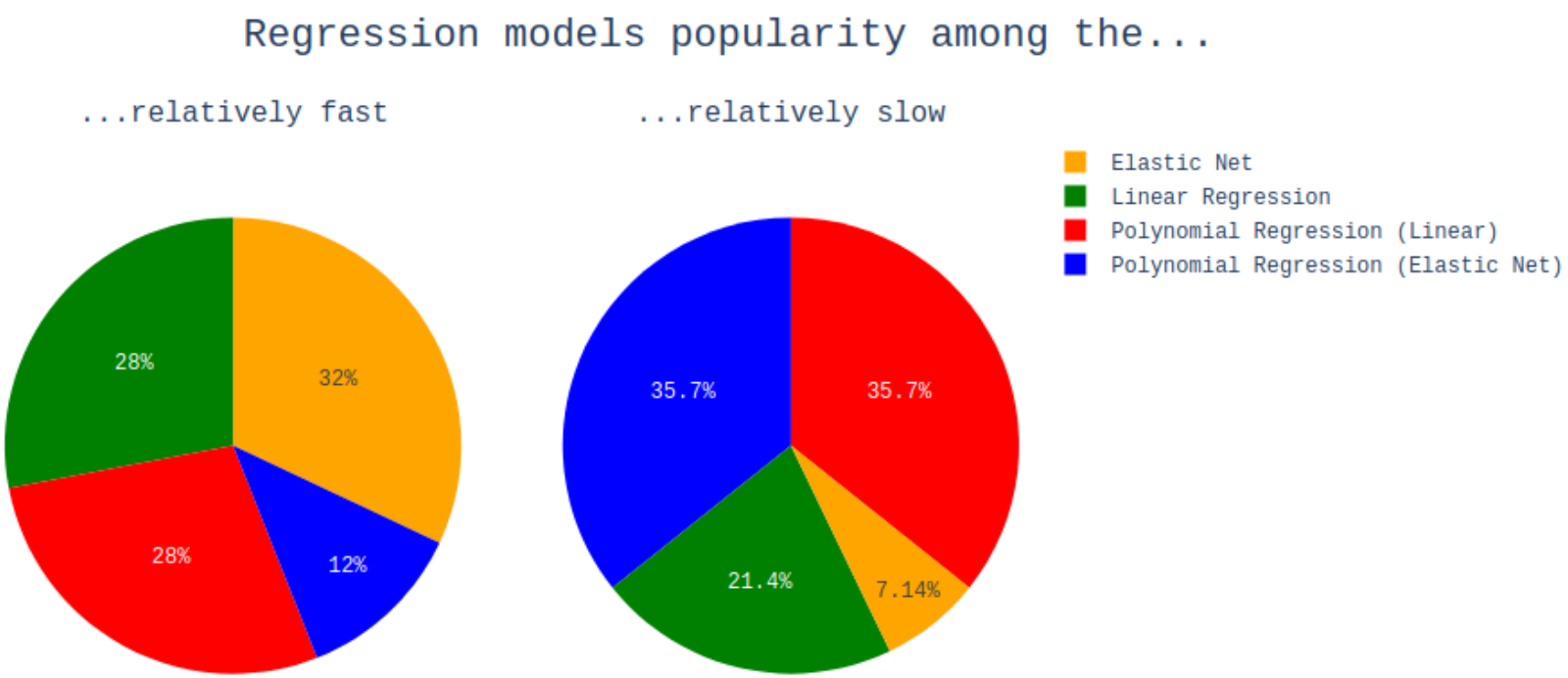
# The design of OCLBoi

- **OCLBoi** (*"OpenCL, But One In-particular"*) is our instcounts model
- **kernel-specific** *(one in particular!)*, **hardware-agnostic**
- predicts **instcounts** based on a **gsize** value
- training and testing on the measurements extracted from Rodinia via oclude
- the training/testing phase results in the selection (based on the $R^2$ score) of one of the following regression strategies:
  1. **Linear regression**
  2. **Elastic Net regression** (i.e. linear regression with L1 and L2 normalization penalties)
  3. **Polynomial regression of degree 2** based on linear regression
  4. **Polynomial regression of degree 2** based on Elastic Net regression

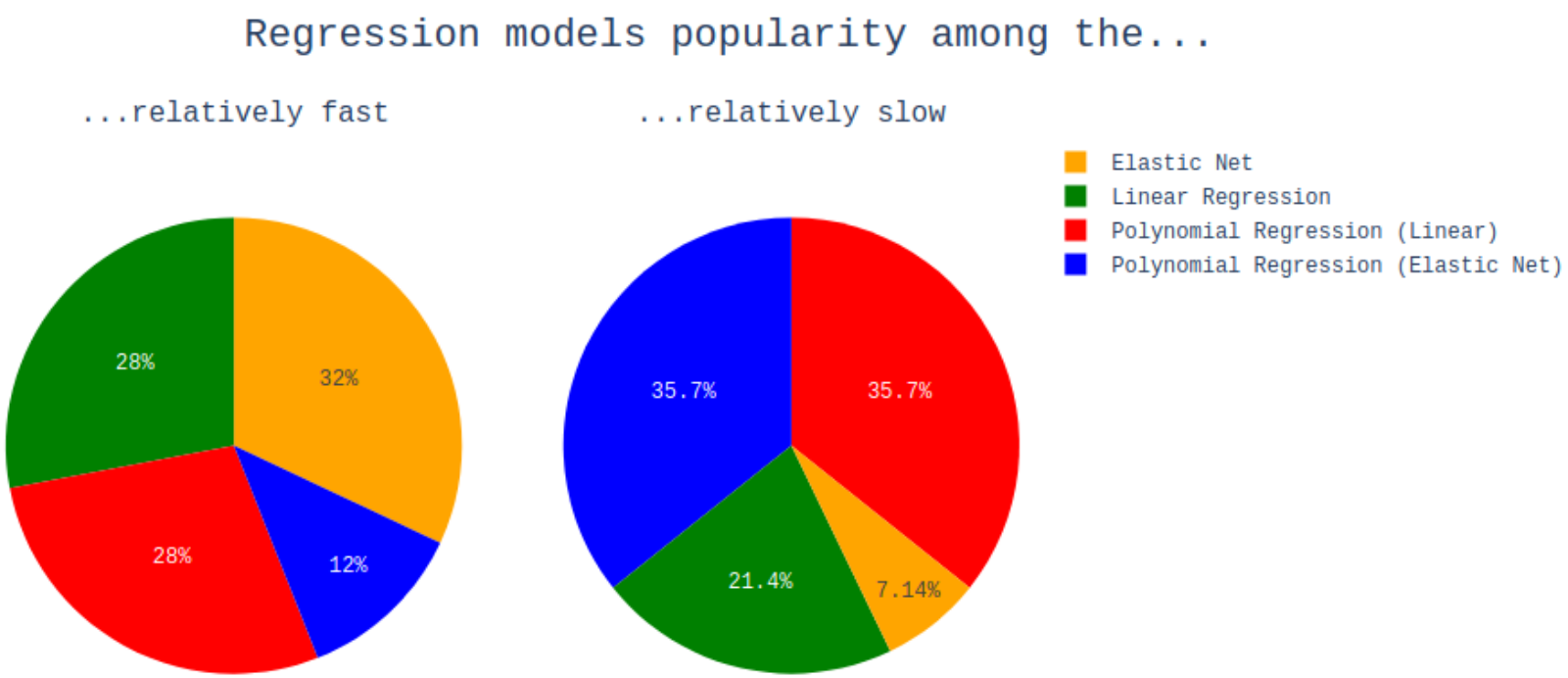
# OCLBoi and the Rodinia Suite



# OCLBoi and the Rodinia Suite

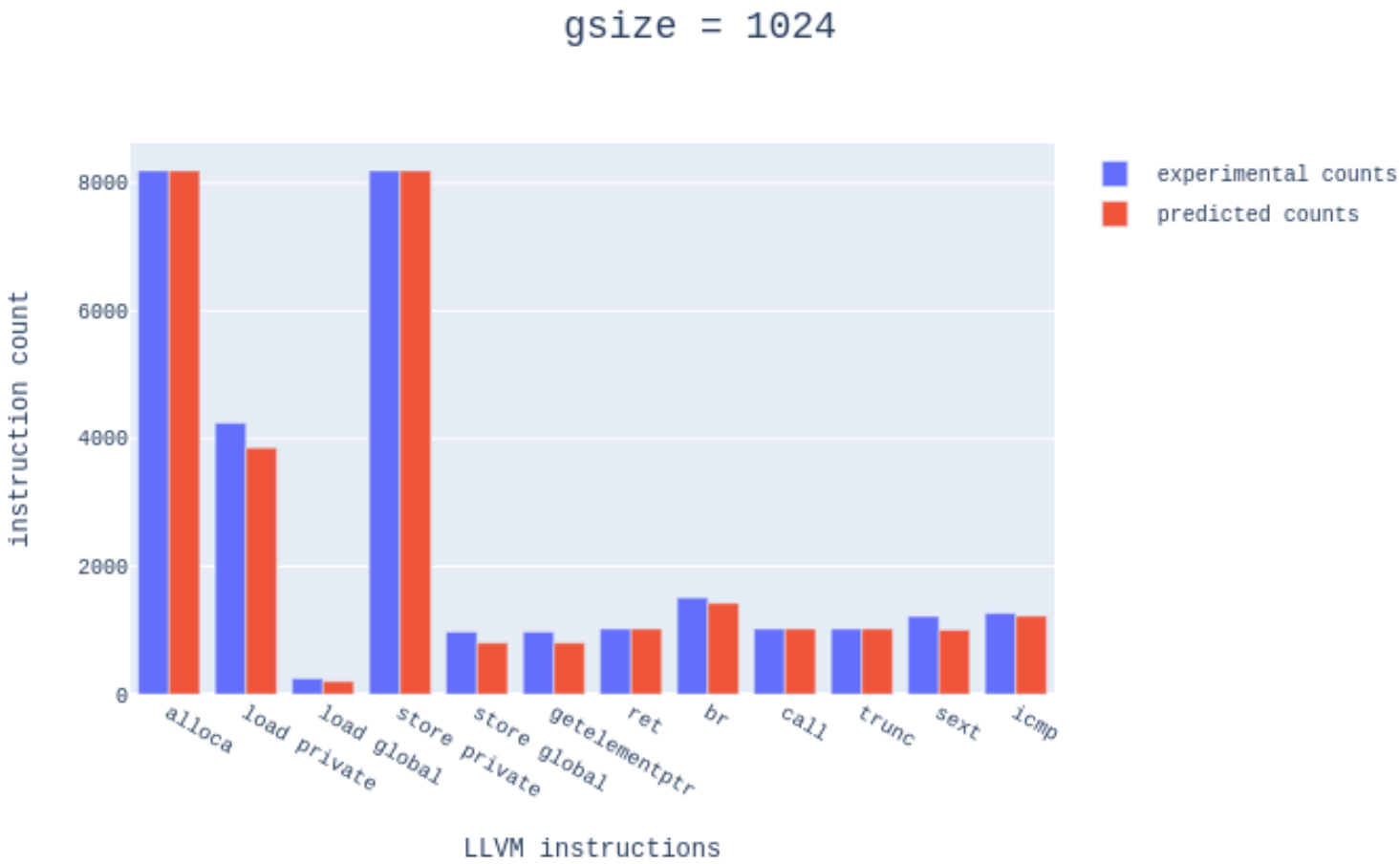


# OCLBoi and the Rodinia Suite

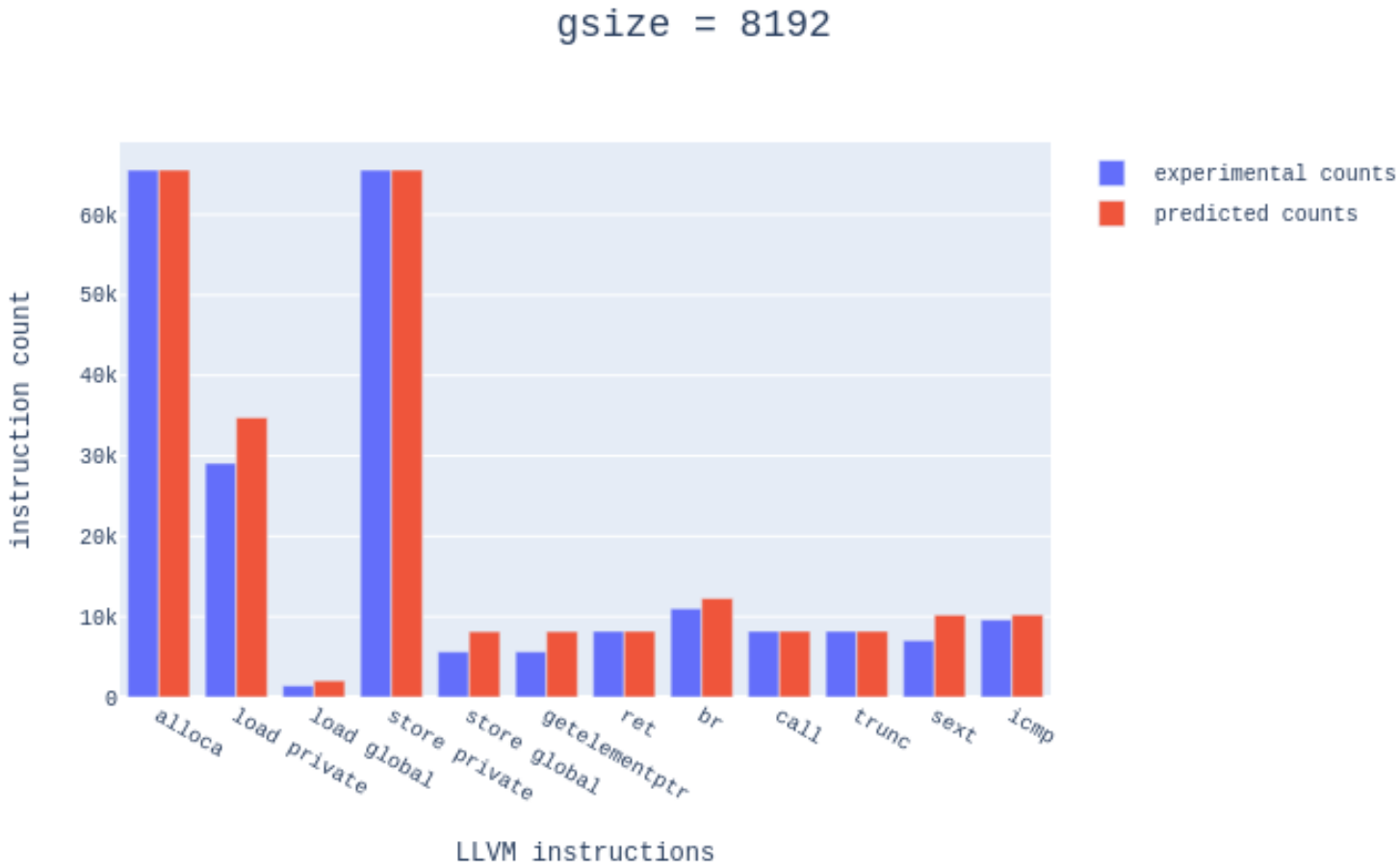



- "relatively fast" $\rightarrow$ **linear models** (60%)
- "relatively slow" $\rightarrow$ **polynomial models** (71.4%)

# OCLBoi in action



# OCLBoi in action



# Now what?

- We have a predictor for the *gsize* $\longmapsto$ *instcounts* relationship

# Now what?

- We have a predictor for the *gsize* $\longmapsto$ *instcounts* relationship
- **What to do with it?**

# Now what?

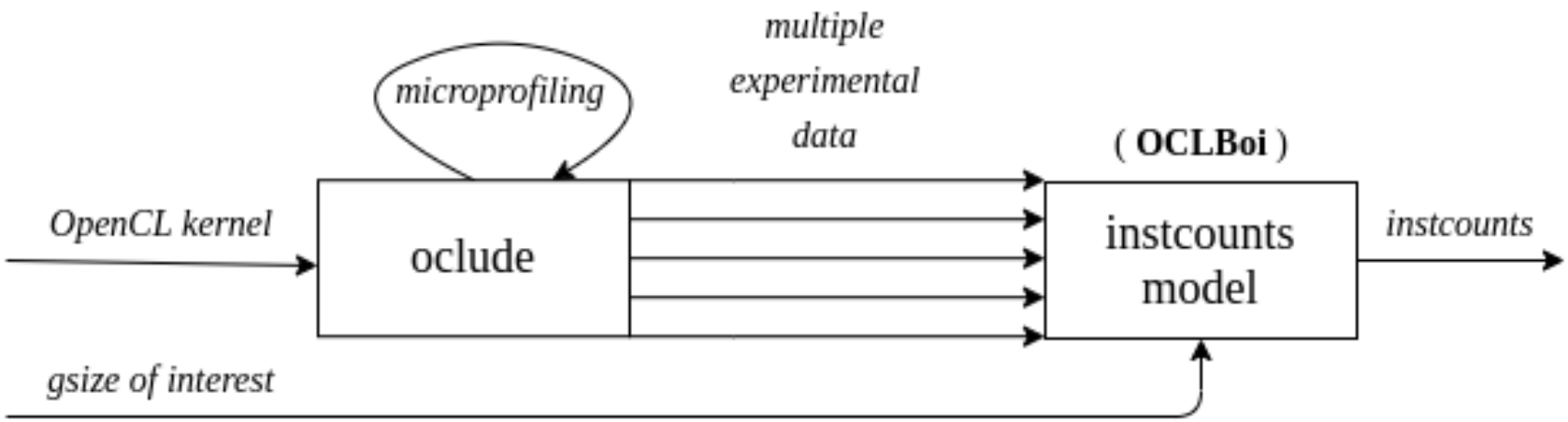
- We have a predictor for the *gsize* $\longmapsto$ *instcounts* relationship
- **What to do with it?**
- How to prove that **it was not all for nothing?**

# Now what?

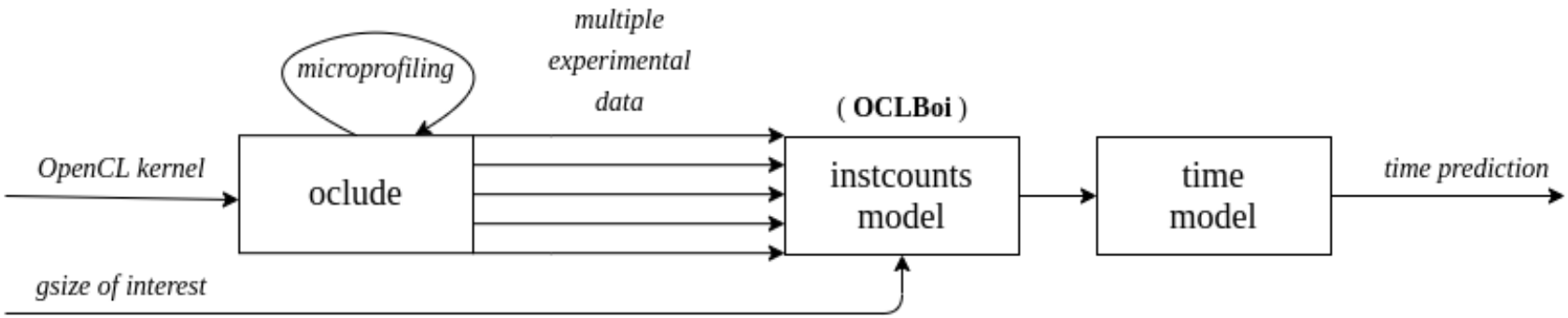
- We have a predictor for the *gsize* $\longmapsto$ *instcounts* relationship
- **What to do with it?**
- How to prove that **it was not all for nothing?**

By predicting execution time!

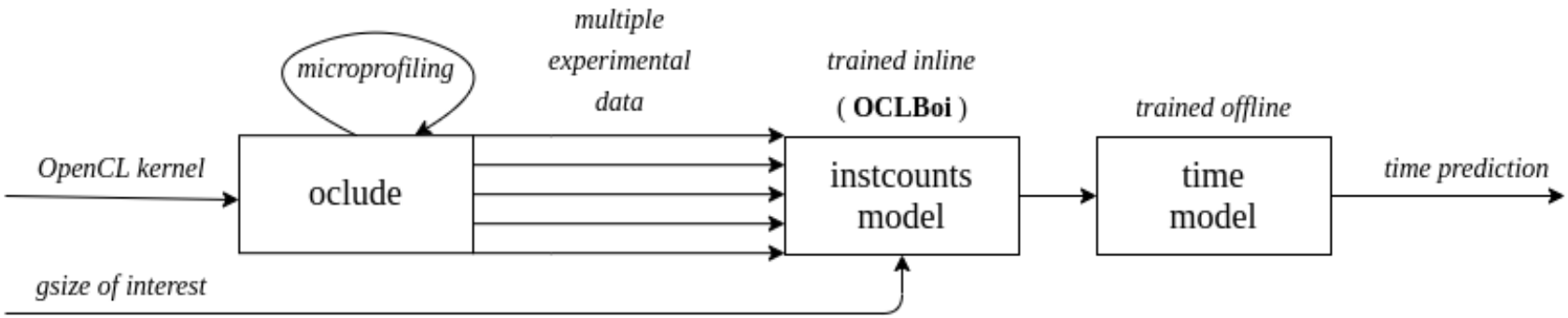
# OCLMan, assemble!



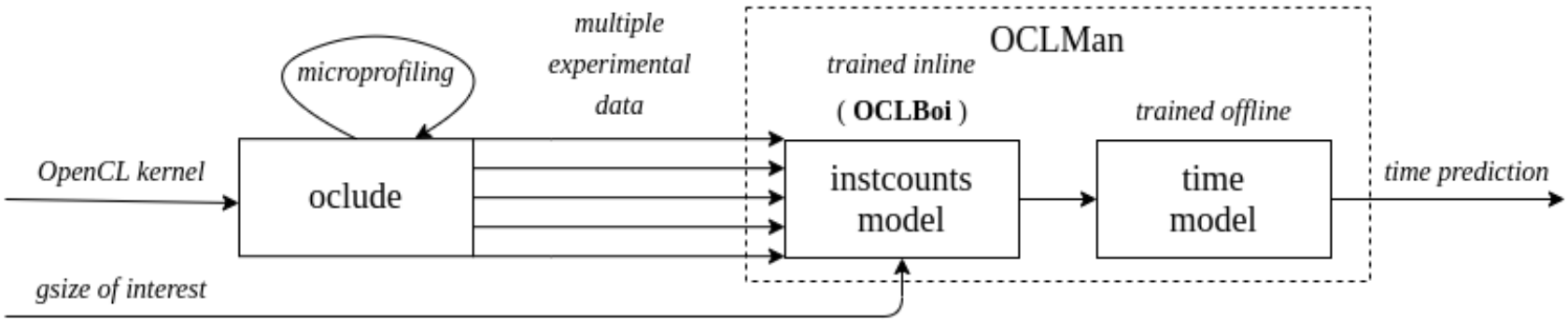
# OCLMan, assemble!



# OCLMan, assemble!



# OCLMan, assemble!



# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the *instcounts* $\longmapsto$ $t_{exec}$ relationship is trained

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**
  - That's it; OCLMan is ready to predict

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**
  - That's it; OCLMan is ready to predict
- **Using OCLMan to predict execution times**

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**
  - That's it; OCLMan is ready to predict
- **Using OCLMan to predict execution times**
  1. A **kernel** and a **gsize** value are provided

# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**
  - That's it; OCLMan is ready to predict
- **Using OCLMan to predict execution times**
  1. A **kernel** and a **gsize** value are provided
  2. A (kernel-specific) OCLBoi is **trained on the fly**

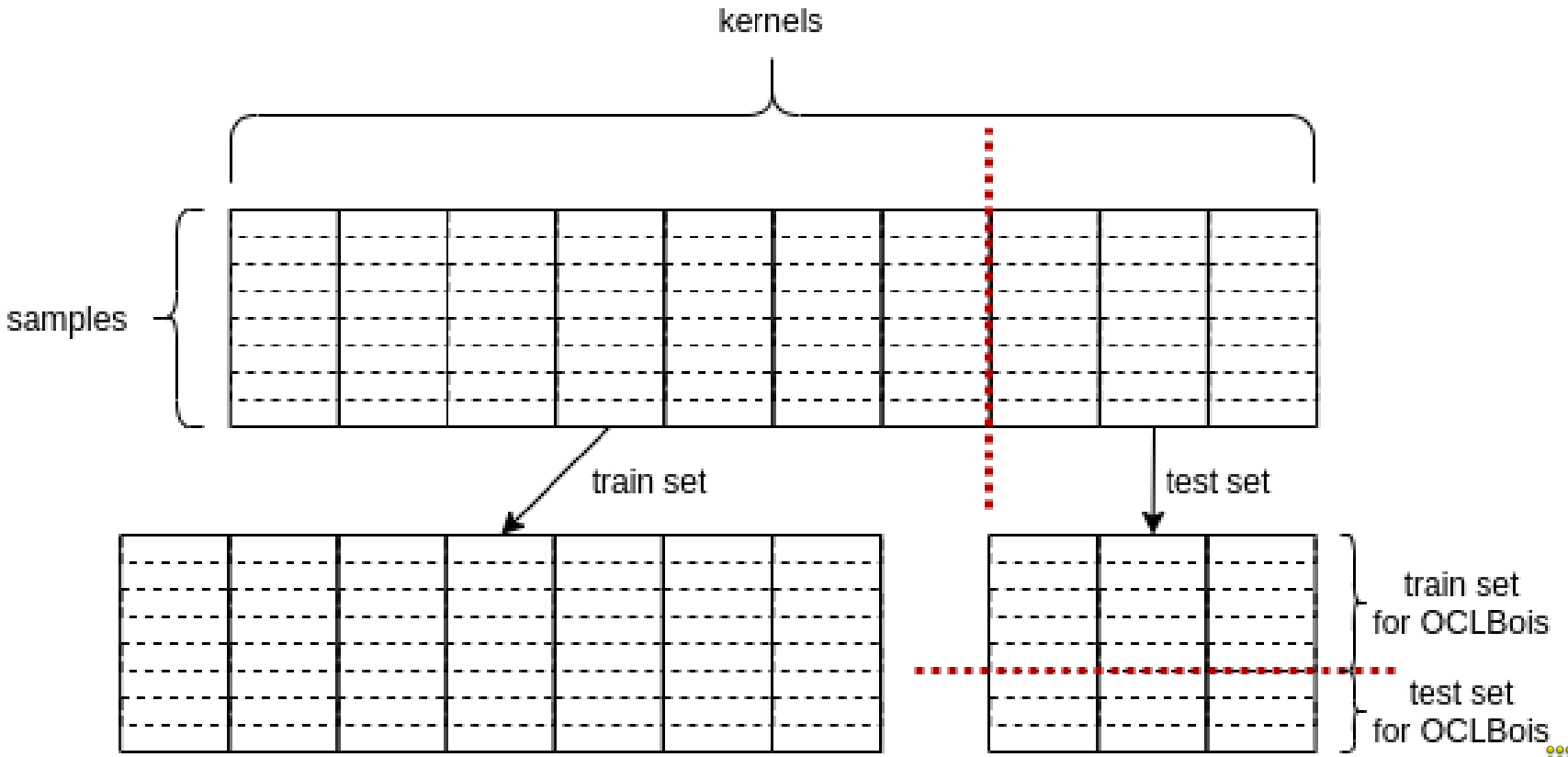
# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**
  - That's it; OCLMan is ready to predict
- **Using OCLMan to predict execution times**
  1. A **kernel** and a **gsize** value are provided
  2. A (kernel-specific) OCLBoi is **trained on the fly**
  3. The input **gsize** value is fed into the pipeline...

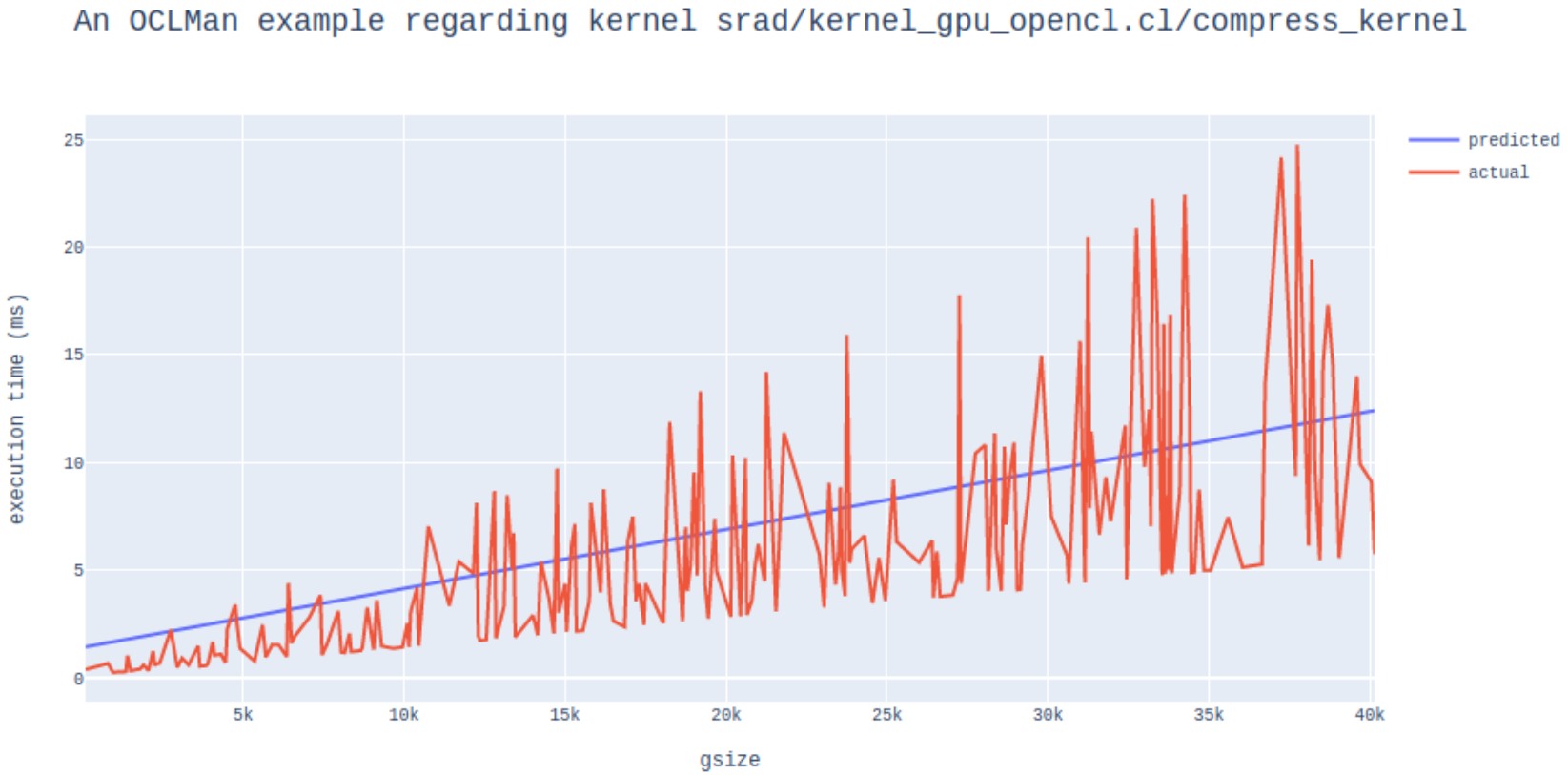
# OCLMan workflow

- **OCLMan** (*"OpenCL Maybe? Approximately? Nope!"*) is our **end-to-end execution time prediction methodology**
- It consists of:
  1. A kernel-specific, hardware-agnostic instcounts model (OCLBoi)
  2. A kernel-agnostic, hardware-specific time model
- **Training OCLMan**
  - A regressor for the $instcounts \longmapsto t_{exec}$ relationship is trained
  - This is the **time model**
  - That's it; OCLMan is ready to predict
- **Using OCLMan to predict execution times**
  1. A **kernel** and a **gsize** value are provided
  2. A (kernel-specific) OCLBoi is **trained on the fly**
  3. The input **gsize** value is fed into the pipeline...
  4. ...and we have a prediction!

# OCLMan training



# OCLMan in action



# The measure of a man

# The measure of a man

- How to evaluate OCLMan?

# The measure of a man

- How to evaluate OCLMan?
- How to know if the dynamic information we extracted was worth it?

# The measure of a man

- How to evaluate OCLMan?
- How to know if the dynamic information we extracted was worth it?
- How to know if we perform better than a static model?

# The measure of a man

- How to evaluate OCLMan?
- How to know if the dynamic information we extracted was worth it?
- How to know if we perform better than a static model?

Let's build one!

# The measure of a man

- How to evaluate OCLMan?
- How to know if the dynamic information we extracted was worth it?
- How to know if we perform better than a static model?

**Let's build one!**

To build it, we will simply **replace dynamic instcounts with the static ones** of the kernel

# Assumptions for OCLBase

# Assumptions for OCLBase

## Assumption 1

$t_{exec}$ is a linear function of instcounts

$$t_{exec} = t_{add} count_{add} + t_{sub} count_{sub} + t_{mul} count_{mul} + \dots$$

# Assumptions for OCLBase

## Assumption 1

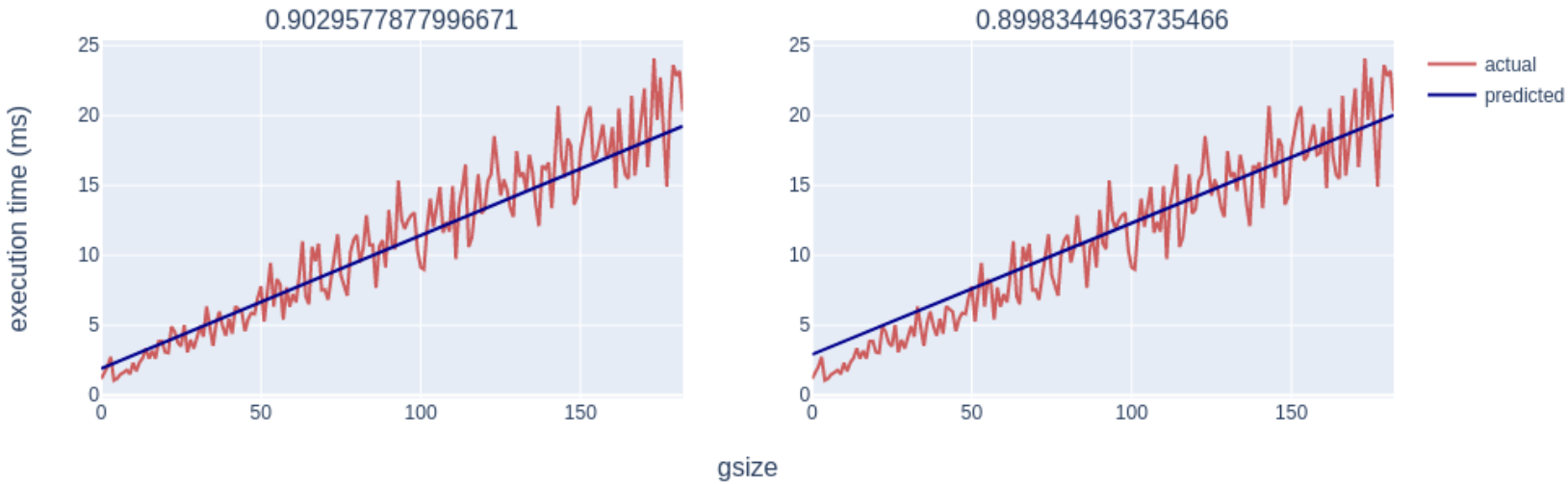
$t_{exec}$ is a linear function of instcounts

$$t_{exec} = t_{add} count_{add} + t_{sub} count_{sub} + t_{mul} count_{mul} + \dots$$

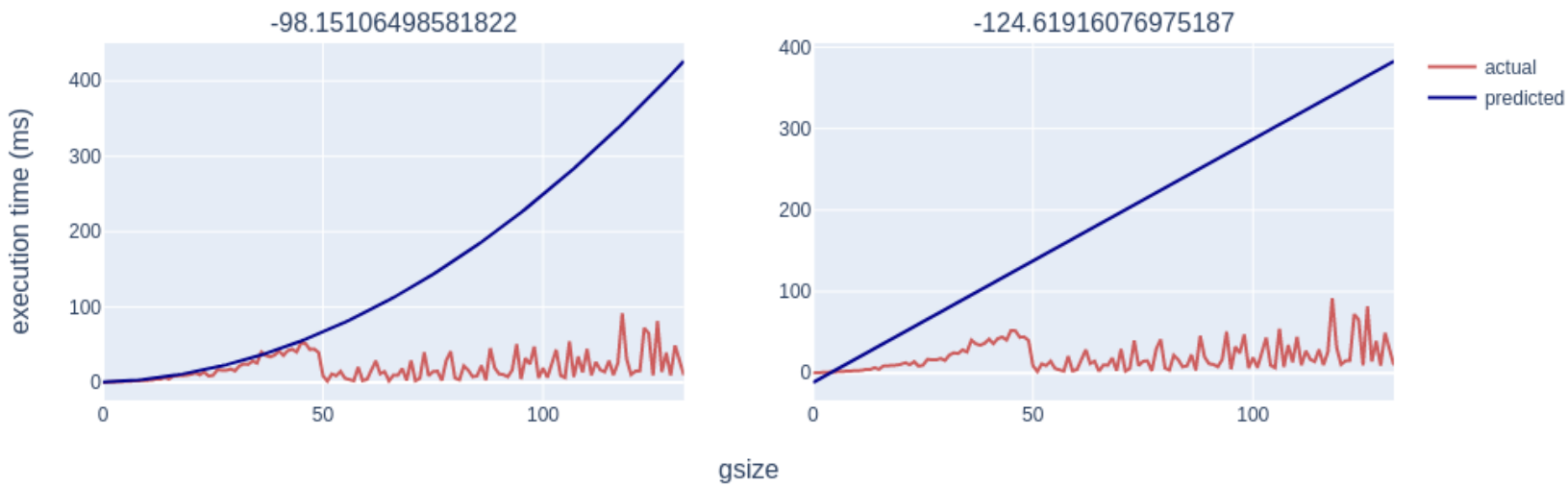
## Assumption 2

$$gsize \longmapsto instcounts : (at\ most)\ polynomial,\ proven$$
$$instcounts \longmapsto t_{exec} : linear,\ assumed$$
$$\Downarrow$$
$$gsize \longmapsto t_{exec} : (at\ most)\ polynomial,\ assumed$$

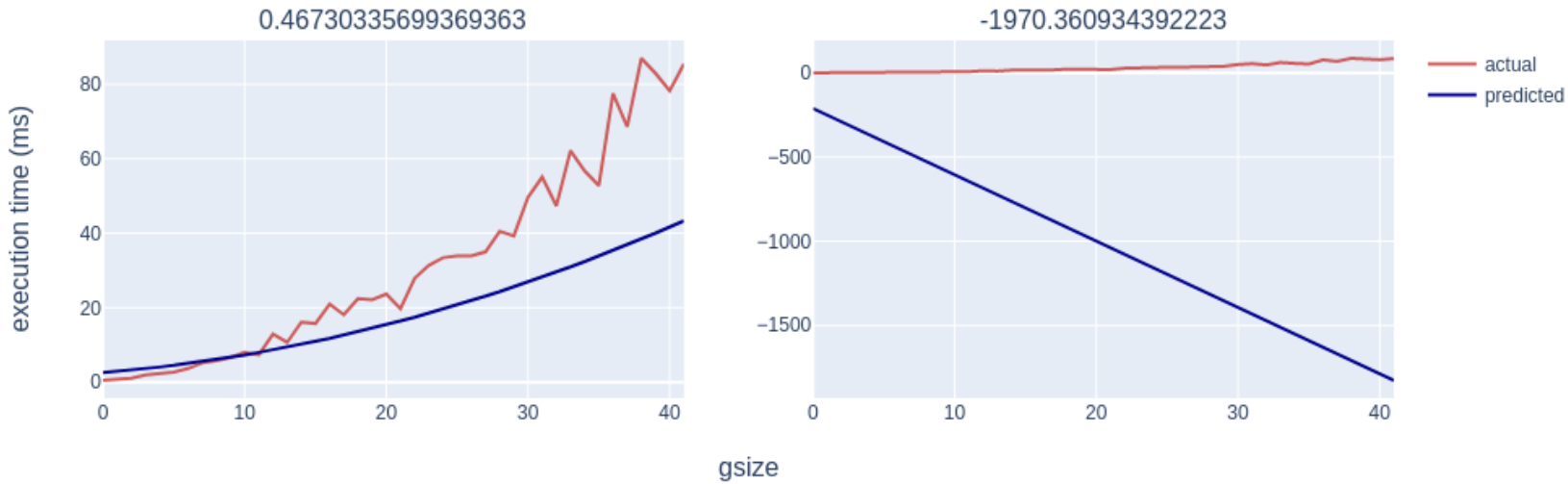
# OCLMan (left) vs. OCLBase (right)



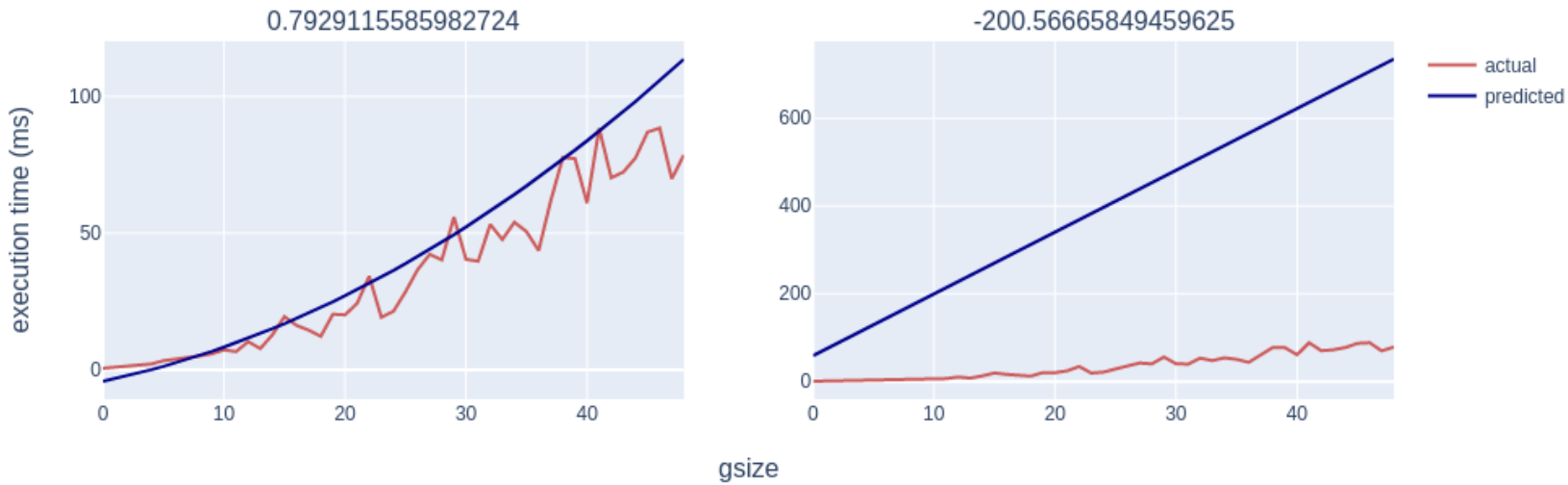
# OCLMan (left) vs. OCLBase (right)



# OCLMan (left) vs. OCLBase (right)



# OCLMan (left) vs. OCLBase (right)



# Final remarks

# Final remarks

- **It was worth it**.

# Final remarks

- **It was worth it**.
- OCLMan was performing **steadily better** no matter the number of times we compared it to OCLBase or the train-test split of the kernels
  - 0.79 vs. -200.56 **(!)**
  - 0.47 vs. -1970.36 **(!!)**

# Final remarks

- **It was worth it**.
- OCLMan was performing **steadily better** no matter the number of times we compared it to OCLBase or the train-test split of the kernels
  - 0.79 vs. -200.56 **(!)**
  - 0.47 vs. -1970.36 **(!!)**
- These results mean that:

# Final remarks

- **It was worth it**.
- OCLMan was performing **steadily better** no matter the number of times we compared it to OCLBase or the train-test split of the kernels
  - 0.79 vs. -200.56 **(!)**
  - 0.47 vs. -1970.36 **(!!)**
- These results mean that:
  1. `oclude` extracts **valuable dynamic information** that surpasses the static approach

# Final remarks

- **It was worth it**.
- OCLMan was performing **steadily better** no matter the number of times we compared it to OCLBase or the train-test split of the kernels
  - 0.79 vs. -200.56 **(!)**
  - 0.47 vs. -1970.36 **(!!)**
- These results mean that:
  1. `oclude` extracts **valuable dynamic information** that surpasses the static approach
  2. `OCLMan` and its `OCLBois` manage to **capture that additional information** and **make something useful out of it**.

# Future work

# Future work

- `oclude` could be re-written to instrument some form of **intermediate representation (IR) code** (e.g. LLVM bitcode) instead of the source code

# Future work

- `oclude` could be re-written to instrument some form of **intermediate representation (IR) code** (e.g. LLVM bitcode) instead of the source code
- turn `OCLMan` from a **methodology** into a **toolkit**.

# Future work

- `oclude` could be re-written to instrument some form of **intermediate representation (IR) code** (e.g. LLVM bitcode) instead of the source code
- turn `OCLMan` from a **methodology** into a **toolkit**. E.g.:
  - test more regression models for the time model component

# Future work

- `oclude` could be re-written to instrument some form of **intermediate representation (IR) code** (e.g. LLVM bitcode) instead of the source code
- turn `OCLMan` from a **methodology** into a **toolkit**. E.g.:
  - test more regression models for the time model component
  - take every new kernel into account (?)

# Future work

- `oclude` could be re-written to instrument some form of **intermediate representation (IR) code** (e.g. LLVM bitcode) instead of the source code
- turn `OCLMan` from a **methodology** into a **toolkit**. E.g.:
  - test more regression models for the time model component
  - take every new kernel into account (?)
  - ...

# Future work

- `oclude` could be re-written to instrument some form of **intermediate representation (IR) code** (e.g. LLVM bitcode) instead of the source code
- turn `OCLMan` from a **methodology** into a **toolkit**. E.g.:
  - test more regression models for the time model component
  - take every new kernel into account (?)
  - ...
- more kernels, more devices

*Thank You!*