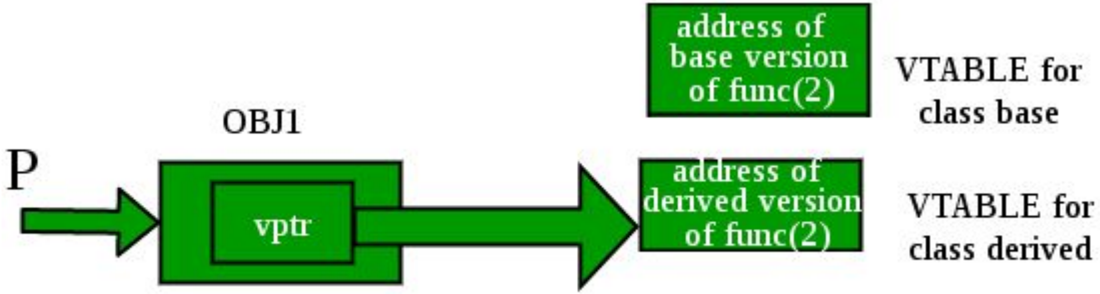
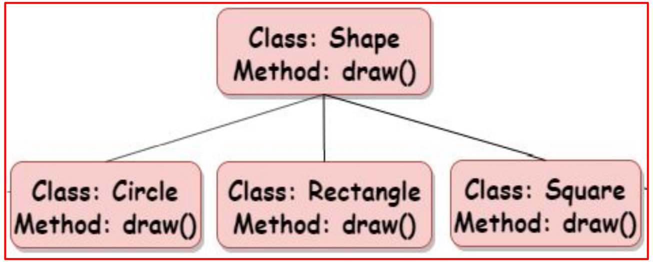
Judging a Type by Its Pointer: Optimizing GPU Virtual Functions

Patrick Xavier Márquez Choque
Jean Carlo Cornejo Cornejo

Introducción



VIRTUAL FUNCTIONS
CALLS IN CPU/GPU



POLYMORPHISM

Problema del Paper

NVIDIA CUDA

AMD

intel

No Research in GPU's Applications

SharedOA Framework

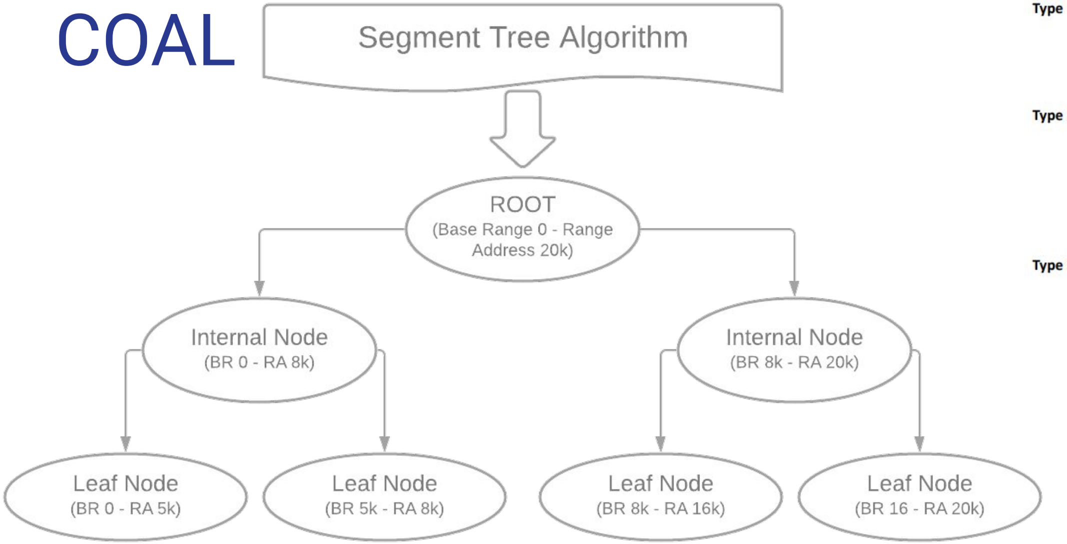
Propuestas del Paper

Coordinated Object Allocation
and function Lookup

&

Hardware Support MMU Mod

COAL



Algorithm 1: Scan algorithm for the virtual range table

Function `ObjectRangeLookup(`*objectAddr*, *funcIndex*`)`

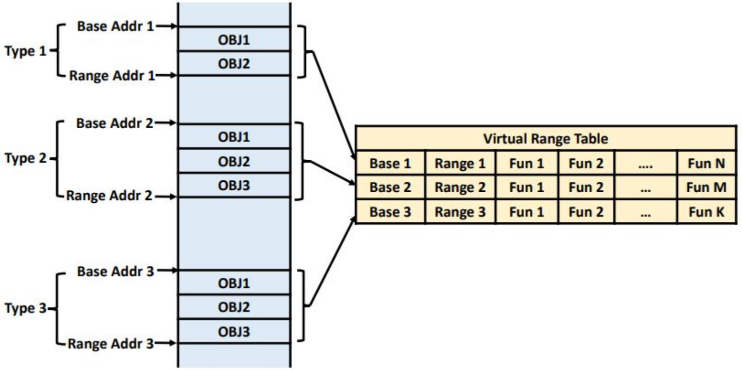
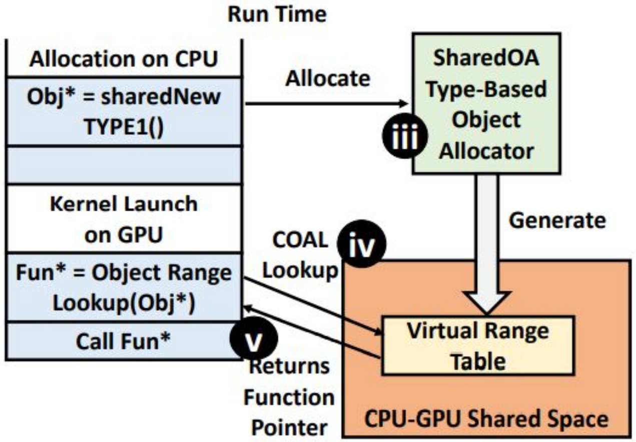
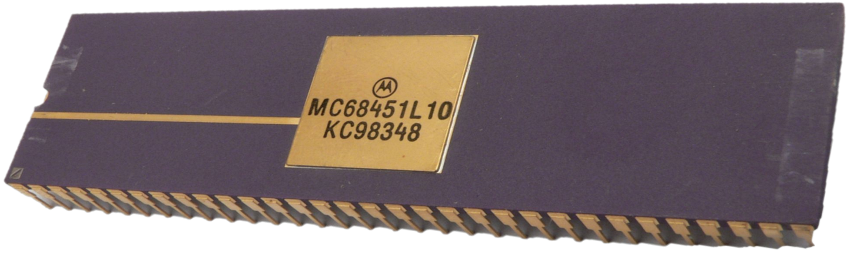


Figure 3: Type-based object allocator.

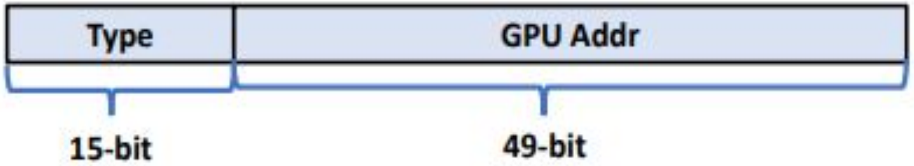


Complexity: $O(\log_2 (K))$

TypePointer



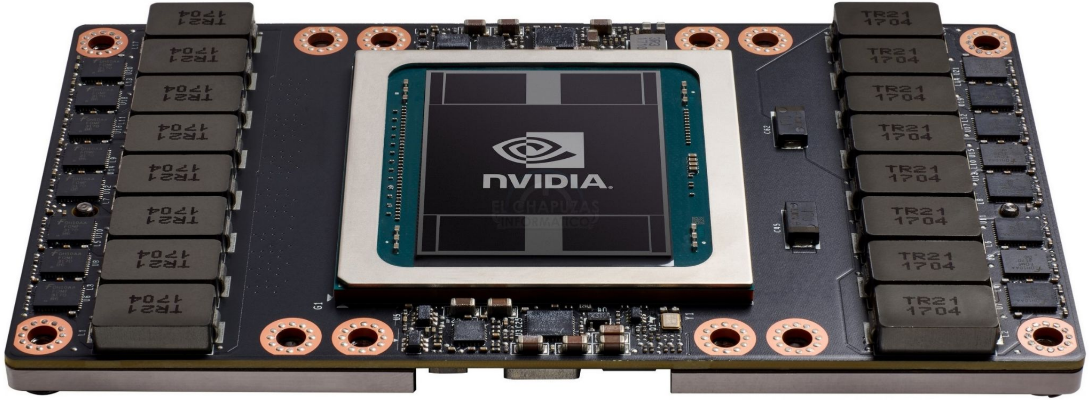
TypePointer Format



1: $SHR \quad Ra, \quad R_{obj}, \quad \#49$
2: $ADD \quad Ra, \quad Ra, \quad R_{vTablesStartAddr}$
3: $LDG \quad Ra, \quad \#vFuncOffset\ (Ra)$
4: $CALL \quad Ra$

(b) Instructions implementing virtual function calls with *TypePointer*

Metodología



TypePointer

GPGPU-SIM

Hardware de Pruebas, con limitación de 4GB como HEAP size máximo

Workloads: Dynasoar, GraphChi-vE, Open Source Ray Tracer

CUDA Concord SharedOA COAL TypePointer

Experimentación

Workload	Description	# Objects	# Types	vFuncs
Dynasoar Workloads [22, 46]				
Traffic (TRAF)	A Nagel-Schreckenberg model traffic simulation to model streets, cars and traffic lights for traffic flows.	1573714	6	74
Game Of Life (GOL)	Game of life is a cellular automaton formulated by John Horton Conway. This benchmark has two abstract classes Cells and Agent.	5645916	4	29
Structure (STUT)	Structure uses the Finite element method to simulate the fracture in a material. The benchmark models the material with springs and nodes.	525000	4	40
Generation (GEN)	Generation is an extension of gol benchmark. The cells in Generation have more intermediate states which lead to more complicated scenarios.	1048576	4	33
GraphChi-vE Workloads [35]				
Breadth First Search (BFS)	BFS traverses graph nodes and updates a level field in a breadth-first manner. The GraphChi-vE BFS implementation defines an abstract class for edges, ChiEdge, and a concrete classEdge, which implements all the virtual functions of ChiEdge.	2254419	4	5
Connected Components (CC)	Connected Component is commonly used for image segmentation and cluster analysis, it employs an iterative node updates according to the labels of adjacent nodes.	2254419	4	6
Page Rank (PR)	Page rank is a classic algorithm to rank the pages of search engine results using iterative updates for each node.	2254419	4	3
GraphChi-vEN Workloads [36]				
Breadth First Search (BFS)	The GraphChi-vEN BFS implementation also defines an abstract base class for vertex, ChiVertex, and a concrete class vertex, which implements ChiVertex's virtual functions.	2254419	4	15
Connected Components (CC)	GraphChi-vEN CC is similar to GraphChi-vE described above. However, GraphChi-vEN CC has both virtual edges and nodes.	2254419	4	15
Page Rank (PR)	GraphChi-vEN PR is similar to GraphChi-vE described above. However, GraphChi-vEN PR has both virtual edges and nodes.	2254419	4	10
Open Source Ray Tracer [40]				
Raytracing (RAY)	RAY performs global rendering of of spheres and planes. The algorithm traces light rays through a scene, bouncing them off of objects, and back to the screen.	1000	3	3

Resultados

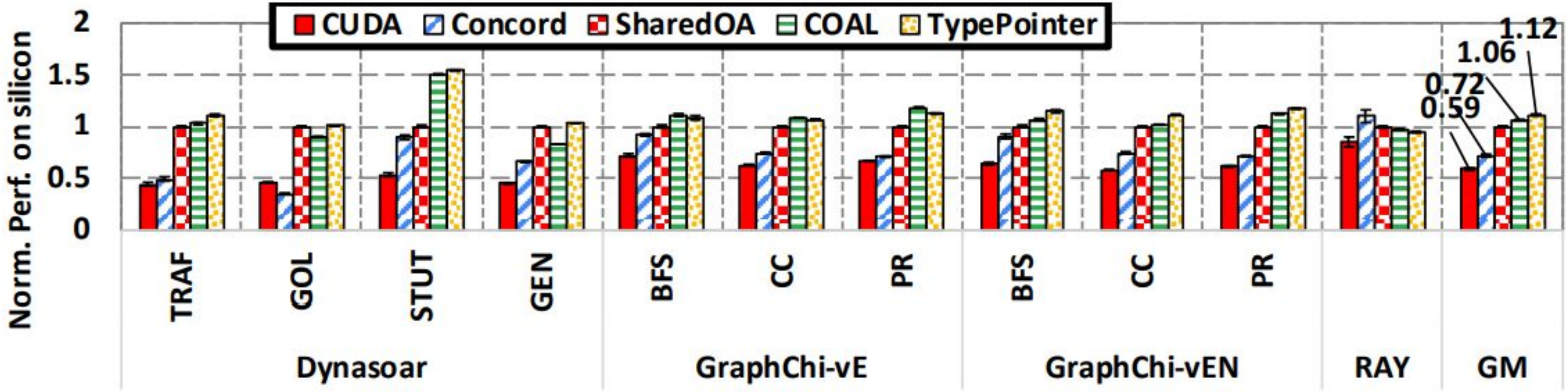


Figure 6: Performance, normalized to *SharedOA* on a silicon V100 GPU, averaged over 10 runs (error-bars=max and min).

Resultados

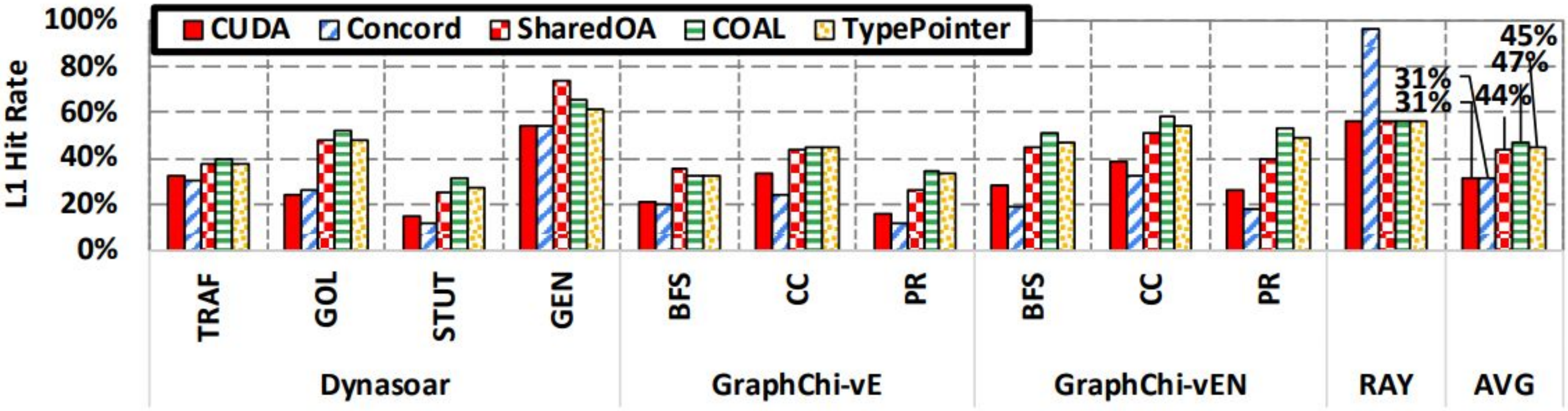


Figure 9: L1 cache hit rate on a silicon V100 GPU.

Conclusiones

NVIDIA CUDA

PERFORMANCE:
80%, 47% & 6% IMPROVEMENT

PERFORMANCE:
90%, 56% & 12%
IMPROVEMENT

GREAT WORK!

CUDA Concord SharedOA COAL TypePointer

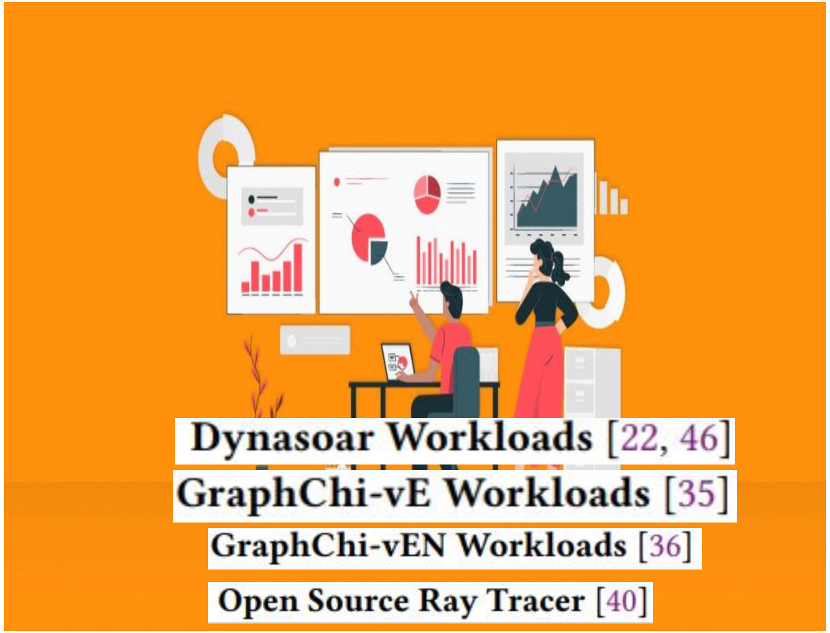
Descripción de las dependencias para las réplicas del trabajo

- **Hardware Dependencies**: Intel x86 machine with NVIDIA Volta architecture GPU with at least 16GB GPU memory. The experiments in this paper use V100 GPU with 32GB GPU memory. 1GB CPU memory is required to contain the evaluation repositories.

- **Software Dependencies**. Ubuntu 18.04 Linux is preferred to run the experiments. NVIDIA CUDA 10.1, 10.2 and 11.1 and GPU driver are required to compile and run the GPU workloads.

Experimentación

Utilización de los Workflow(Benchmarking):



Experimentos con ejemplos simples:

SharedOA, COAL and TypePointer

Software prerequisite

- Ubuntu 18.04.5 LTS Linux
- git
- Python 2.7
- CUDA 10.1: https://developer.nvidia.com/cuda-10.1-download-archive-base
- Transform script for COAL and TypePointer: https://github.com/brad-mengchi/asplos_2021_ae

Experimentación con Ejemplos Simples (COAL)

```
#include "COAL.h"
#define NUM_OBJ 512
class S1 {
public:
  int var;
  __host__ __device__ S1() {}
  virtual __host__ __device__ void inc() = 0;
  virtual __host__ __device__ void dec() = 0;
};

class S2 : public S1 {
public:
  __host__ __device__ S2() {}
  __host__ __device__ void inc() { this->var += 2; }

  __host__ __device__ void dec() { this->var -= 2; }
};
```

```
__global__ void kernel(S1 **ptr) {
  int tid = threadIdx.x + blockDim.x * blockIdx.x;
  // this variable must be defined in every kerenl that uses COAL
  void **vtable;
  if (tid < NUM_OBJ) {
    COAL_S1_inc(ptr[tid]);
    ptr[tid]->inc();
  }
}
```

```
    my_obj_alloc.toDevice();
    int blockSize = 256;
    int numBlocks = (NUM_OBJ + blockSize - 1) / blockSize;
    kernel<<<numBlocks, blockSize>>>(ptr);
    cudaDeviceSynchronize();
```

Réplica del Experimento en COAL

Réplica con el Cluster de Google Colab:
[Google Colab Experiment]

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 465.27       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
| N/A   55C    P8    10W /  70W |      0MiB / 15109MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2020 NVIDIA Corporation
Built on Wed_Jul_22_19:09:09_PDT_2020
Cuda compilation tools, release 11.0, V11.0.221
Build cuda_11.0_bu.TC445_37.28845127_0
```

```
vtable [252][0]:0x8
vtable [252][1]:0x10
Objects Creation Done
Host Call Done
Device Call Done
Host Call Done
Device Call Done
ptr[0].var = 0
ptr[1].var = 0
ptr[2].var = 0
ptr[3].var = 0
```

...   .

```
ptr[505].var = 0
ptr[506].var = 0
ptr[507].var = 0
ptr[508].var = 0
ptr[509].var = 0
ptr[510].var = 0
ptr[511].var = 0
```

```
Type#0:
Type Size: 16     Number of Buckets : 1    Range Size: 1048576     Number of Objs : 512

Avg Types Size 16.000000
```

Experimentación con Ejemplos Simples (SharedOA)

```
#include "mem_alloc.h"
#define NUM_OBJ 512
class S1 {
public:
  int var;
  __host__ __device__ S1() {}
  virtual __host__ __device__ void inc() = 0;
  virtual __host__ __device__ void dec() = 0;
};

class S2 : public S1 {
public:
  __host__ __device__ S2() {}
  __host__ __device__ void inc() { this->var += 2; }

  __host__ __device__ void dec() { this->var -= 2; }
};
```

```
__global__ void kernel(S1 **ptr) {
  int tid = threadIdx.x + blockDim.x * blockIdx.x;
  if (tid < NUM_OBJ)
    ptr[tid]->inc();
}
```

```
my_obj_alloc.toDevice();
int blockSize = 256;
int numBlocks = (NUM_OBJ + blockSize - 1) / blockSize;
kernel<<<numBlocks, blockSize>>>(ptr);
cudaDeviceSynchronize();
printf("Device Call Done\n");
```

Réplica del Experimento en SharedOA

Réplica en Google Colab:

```
vtable [252][0]:0x8
vtable [252][1]:0x10
Objects Creation Done
Host Call Done
Device Call Done
Host Call Done
Device Call Done
ptr[0].var = 0
ptr[1].var = 0
ptr[2].var = 0
ptr[3].var = 0
ptr[4].var = 0
ptr[5].var = 0
ptr[6].var = 0
ptr[7].var = 0
ptr[8].var = 0
ptr[9].var = 0
```

```
ptr[508].var = 0
ptr[509].var = 0
ptr[510].var = 0
ptr[511].var = 0
```

```
Type#0:
Type Size: 16    Number of Buckets : 1   Range Size: 1048576    Number of Objs : 512

Avg Types Size 16.000000
```

Experimentación con Ejemplos Simples (TP)

```
#include "mem_alloc_tp.h"
#include "TP.h"
#define NUM_OBJ 512
class S1 {
public:
  int var;
  __host__ __device__ S1() {}
  virtual __host__ __device__ void inc() = 0;
  virtual __host__ __device__ void dec() = 0;
};

class S2 : public S1 {
public:
  __host__ __device__ S2() {}
  __host__ __device__ void inc() { this->var += 2; }

  __host__ __device__ void dec() { this->var -= 2; }
};
```

```
__global__ void kernel(S1 **ptr) {
  int tid = threadIdx.x + blockDim.x * blockIdx.x;
  // this variable must be defined in every kerenl that uses COAL
  void **vtable;
  if (tid < NUM_OBJ) {
    S1 * obPtr = ptr[tid];
    TP_S1_inc(obPtr);
    CLEANPTR(obPtr,S1 *)->inc();
  }
}
```

```
my_obj_alloc.toDevice();
int blockSize = 256;
int numBlocks = (NUM_OBJ + blockSize - 1) / blockSize;
kernel<<<numBlocks, blockSize>>>(ptr);
cudaDeviceSynchronize();
```

Réplica del Experimento en TP

Réplica en Google Colab:

```
vtable [252][0]:0x8
vtable [252][1]:0x10
total_count 1048576 , total 1048576 , type_size 16
Objects Creation Done
Type0:  (nil)
Type1:  0x8
#############
gpuptr 0x7f78dc001138
Device Call Done
ptr[0].var = 2
ptr[1].var = 2
```

```
ptr[510].var = 2
ptr[511].var = 2
Type#0:
Type Size: 16    Number of Buckets : 1    Range Size: 1048576     Number of Objs : 512

Avg Types Size 16.000000
```

Link del Repositorio

Github

Paper