GYAN: Accelerating Bioinformatics tools in Galaxy with GPU-Aware Computation Mapping

HiCOMB 2021
20th IEEE International Workshop on High Performance Computational Biology

Authors: Gulsum Gudukbay, Jashwant Raj Gunasekaran, Yilin Feng, Mahmut T. Kandemir, Chita R. Das, Anton Nekrutenko, Paul Medvedev, Bjorn Gruning, Nate Coraor, Enis Afgan, Nathan Roach

Overview

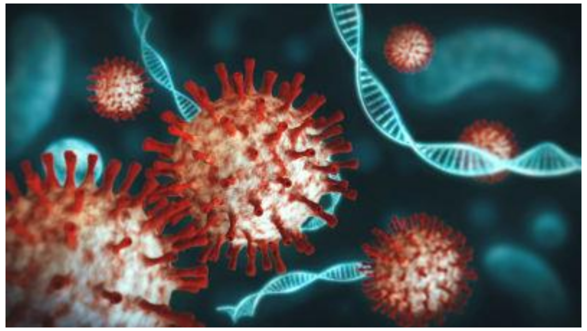
Overview

- Introduction
- Motivation
- Design of GYAN
- Implementation Details
- Evaluation
- Conclusion

Introduction - Galaxy Prominence

Source: https://galaxyproject.org/use/

Introduction - Accelerating Bioinformatics Applications



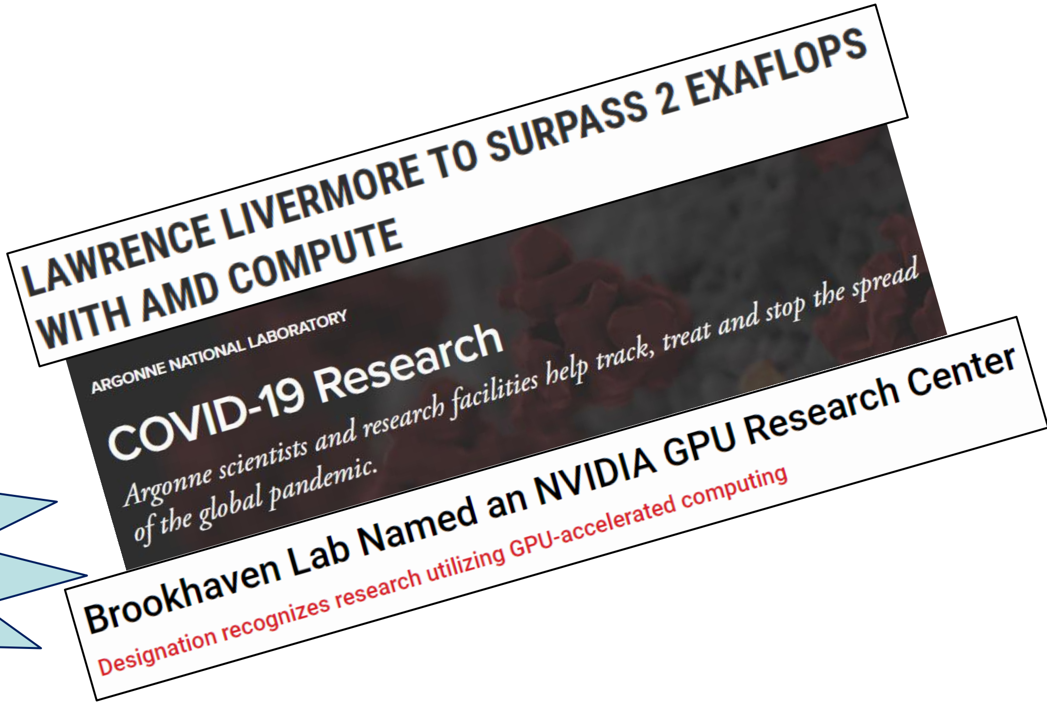
COVID-19 Genome Sequencing & Vaccine discovery

With GPU support, hundreds of thousands of experiments for various important domains will be accelerated within Galaxy, allowing faster and higher bandwidth research.

Direct Coulomb Summation: **~45x**
Racon: **~2x**
Bonito: **~50x**

COVID-19 Vaccine Study: 5X Speedup

pyPaSWAS: 33X Speedup



LAWRENCE LIVERMORE TO SURPASS 2 EXAFLOPS WITH AMD COMPUTE

ARGONNE NATIONAL LABORATORY

COVID-19 Research

Argonne scientists and research facilities help track, treat and stop the spread of the global pandemic.

Brookhaven Lab Named an NVIDIA GPU Research Center

Designation recognizes research utilizing GPU-accelerated computing

Overview

Motivation

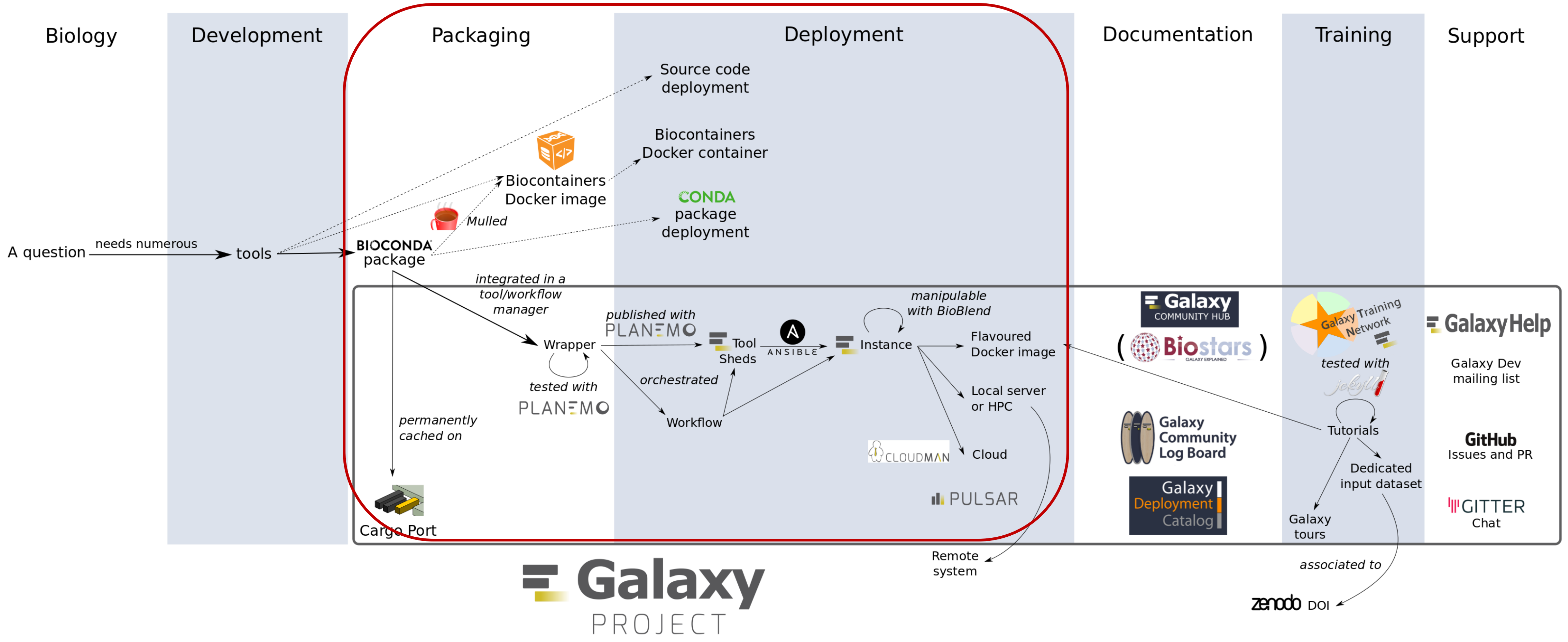
- Current implementation of Galaxy does not allow GPU-enabled tools
- It is non-trivial to integrate GPUs into the current Galaxy framework without affecting the original user experience.

Overview

GYAN

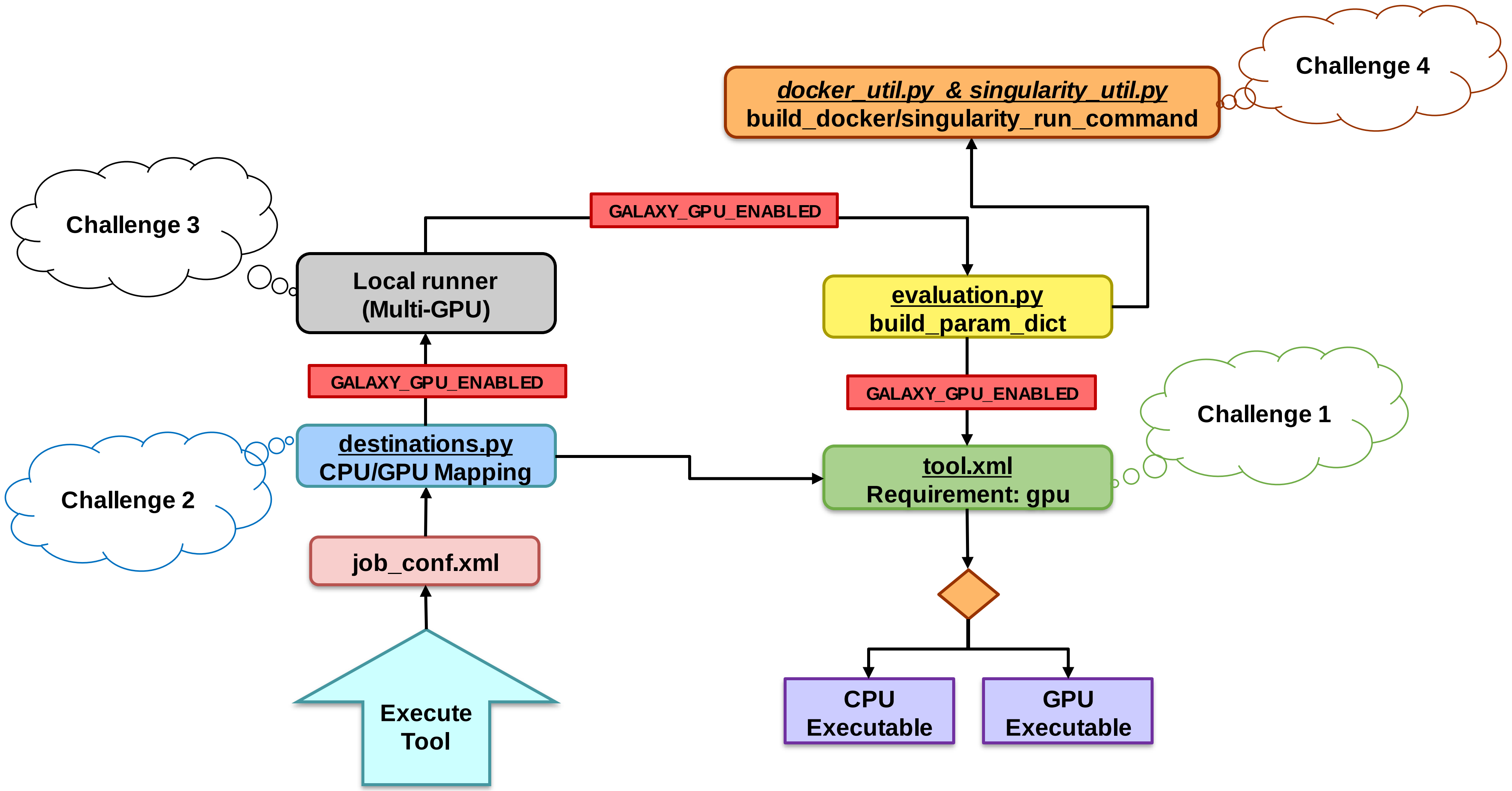
- Make Galaxy GPU-aware
  - GPU-supported tools can seamlessly execute in Galaxy
- Intelligent GPU-aware orchestration policy
- Multi-GPU support
  - GPU selection based on the availability/utilization of all GPUs.
- Evaluated GPU support using Racon and Bonito tools.
  - ~2x speedup for Racon and ~50x for Bonito over CPU-only versions

Galaxy End-to-End Architecture



Source: https://galaxyproject.org/develop/architecture/

Overall System Flow



Overview

Implementation Details

1

```
<macros>
  <xml name="requirements">
    <requirements>
      <requirement type="package" version="@VERSION@
  ">racon</requirement>
      <requirement type="compute">gpu</requirement>
    </requirements>
```

The **“compute”** requirement of type **“gpu”** is specified. Allows Galaxy to recognize that the tool requires GPU.

2

```
def dynamic_fun(tool,job):
    flag = 0
    os.environ['GALAXY_GPU_ENABLED'] = "false"

    if tool:
        reqmnts = tool.requirements
        for req in reqmnts:
            if req.type == "compute" and req.name ==
  "gpu":
                flag = 1
        if gpu_count > 0 and flag == 1:
            os.environ['GALAXY_GPU_ENABLED'] = "true
  "
            return "local_gpu"
        else:
            os.environ['GALAXY_GPU_ENABLED'] = "
  false"
            return "local_cpu"
```

Environment variable ‘GALAXY_GPU_ENABLED’ introduced to Galaxy backend in a new job rule.

3

```
<job_conf>
  <plugins>
    ...
    <plugin id="dynamic" type="runner">
      <param id="rules_module">galaxy.jobs.rules</
    param>
    </plugin>
  </plugins>
  <destinations default="gpu_cpu_decision">
    <destination id="gpu_cpu-decision" runner="
    dynamic">
      <param id="type">python</param>
      <param id="function">dynamic_map</param>
```

New job rule used in job configuration file.

```
def build_param_dict(self, incoming, input_datasets,
     output_datasets, output_collections):
    ...
    param_dict["input"]  = input
    ...
    param_dict['__galaxy_gpu_enabled__'] = os.
    environ['GALAXY_GPU_ENABLED']
    param_dict.update(self.tool.
    template_macro_params)
    # All parameters go into the param_dict
    param_dict.update(incoming)
```

4

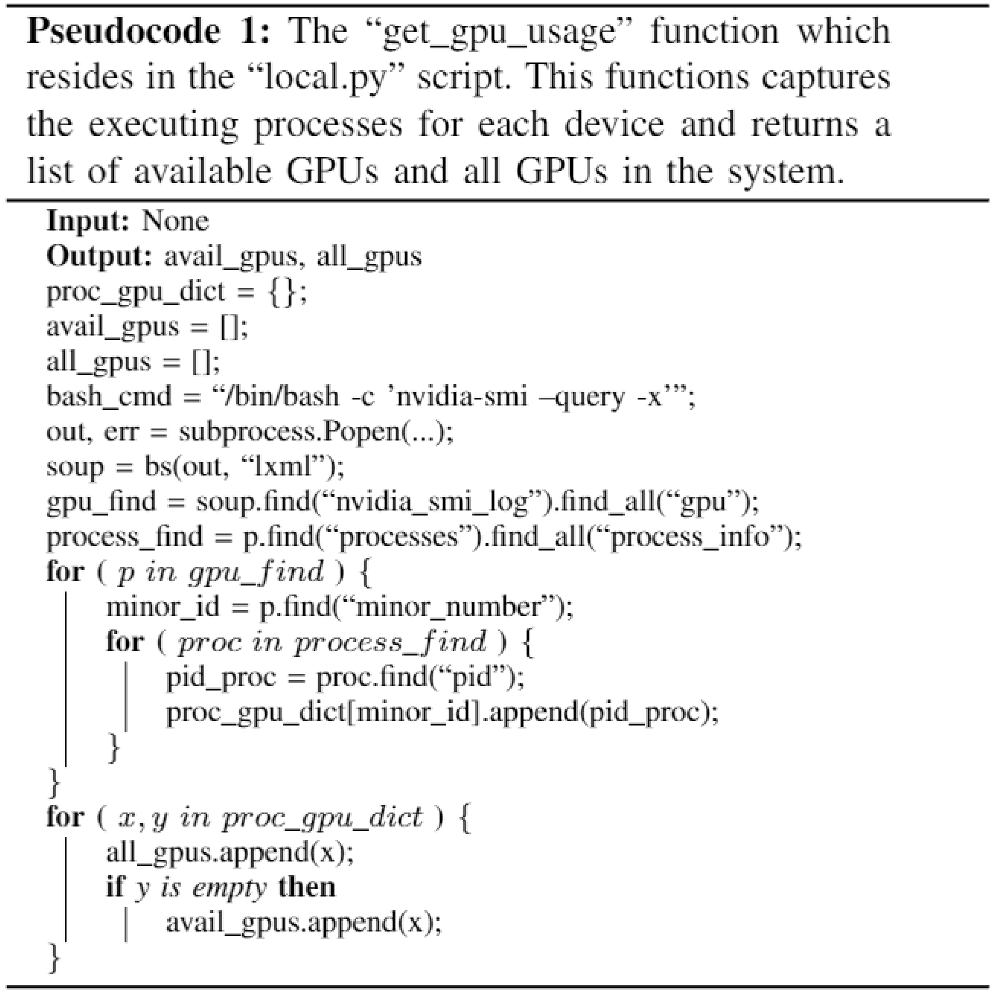
Environment variable ‘GALAXY_GPU_ENABLED’ is exposed to tool configuration file using the parameter dictionary.

5

```
<tool id="racon" name="Racon" version="@VERSION@">
  ...
  <command detect_errors="exit_code"><![CDATA[
    ...
    #if $__galaxy_gpu_enabled__=="true":
      racon_gpu
    #else:
      racon_cpu
    #end if#
    ...
  ]]></command>
</tool>
```

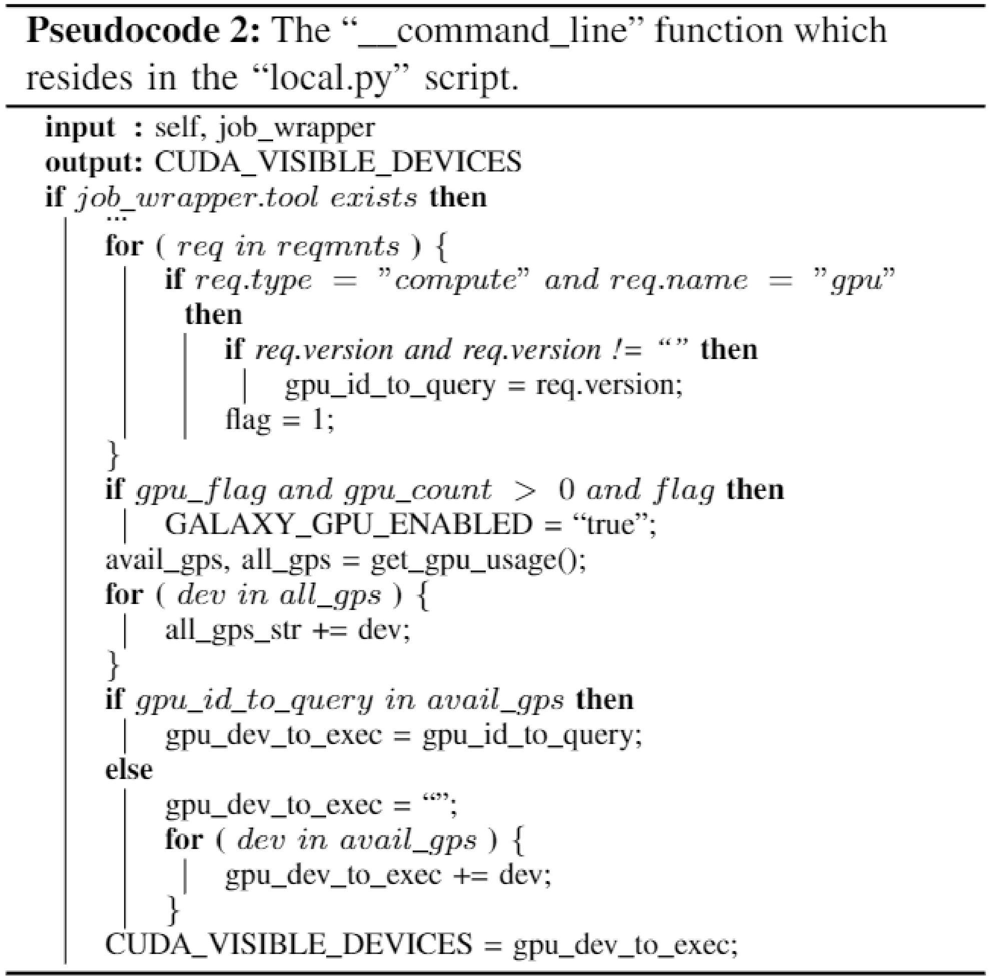
After the exposure is done, the job rule sets the value for $GALAXY_GPU_ENABLED and uses the value of it inside the tool wrapper file in order to select which executable to use. It is also utilized when deciding on which parameters to use later.

Implementation Details: Multi-GPU



Pseudocode 1: The "get_gpu_usage" function which resides in the "local.py" script. This functions captures the executing processes for each device and returns a list of available GPUs and all GPUs in the system.

```
Input: None
Output: avail_gpus, all_gpus
proc_gpu_dict = {};
avail_gpus = [];
all_gpus = [];
bash_cmd = "/bin/bash -c 'nvidia-smi –query -x'";
out, err = subprocess.Popen(...);
soup = bs(out, "lxml");
gpu_find = soup.find("nvidia_smi_log").find_all("gpu");
process_find = p.find("processes").find_all("process_info");
for ( $p\ in\ gpu\_find$ ) {
    minor_id = p.find("minor_number");
    for ( $proc\ in\ process\_find$ ) {
        pid_proc = proc.find("pid");
        proc_gpu_dict[minor_id].append(pid_proc);
    }
}
for ( $x, y\ in\ proc\_gpu\_dict$ ) {
    all_gpus.append(x);
    if y is empty then
        avail_gpus.append(x);
}
```



Pseudocode 2: The "__command_line" function which resides in the "local.py" script.

```
input : self, job_wrapper
output: CUDA_VISIBLE_DEVICES
if $job\_wrapper.tool\ exists$ then
    ...
    for ( $req\ in\ reqmnts$ ) {
        if $req.type = "compute"\ and\ req.name = "gpu"$
         then
            if req.version and req.version != "" then
                gpu_id_to_query = req.version;
            flag = 1;
    }
    if $gpu\_flag\ and\ gpu\_count > 0\ and\ flag$ then
        GALAXY_GPU_ENABLED = "true";
    avail_gps, all_gps = get_gpu_usage();
    for ( $dev\ in\ all\_gps$ ) {
        all_gps_str += dev;
    }
    if $gpu\_id\_to\_query\ in\ avail\_gps$ then
        gpu_dev_to_exec = gpu_id_to_query;
    else
        gpu_dev_to_exec = "";
        for ( $dev\ in\ avail\_gps$ ) {
            gpu_dev_to_exec += dev;
        }
    CUDA_VISIBLE_DEVICES = gpu_dev_to_exec;
```

Implementation Details: Containerized Tools

- Galaxy does not launch the containers with GPU support.
- We added this support by adding flags to the container launch script.

```
def build_docker_run_command(...):
    ...
    if run_extra_arguments:
        command_parts.append(run_extra_arguments)
    if os.environ['GALAXY_GPU_ENABLED'] == "true":
        command_parts.append("--gpus all")
```

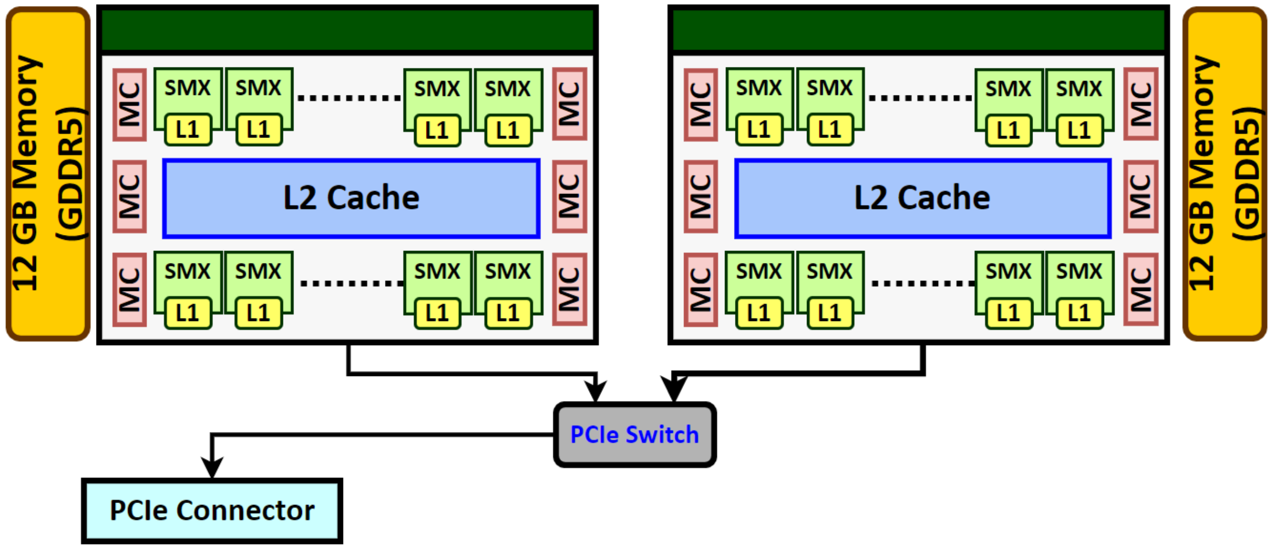
```
def build_singularity_run_command(...):
    ...
    if run_extra_arguments:
        command_parts.append(run_extra_arguments)
    if os.environ['GALAXY_GPU_ENABLED'] == "true":
        command_parts.append("--nv")
```

Overview

- Introduction
- Motivation
- Design of GYAN
- Implementation Details
- Evaluation
- Conclusion

Experimental Configuration

- 2 NVIDIA Tesla K80 GPUs
- 32 Warp Size
- CUDA-10.2, Python-3.6.9

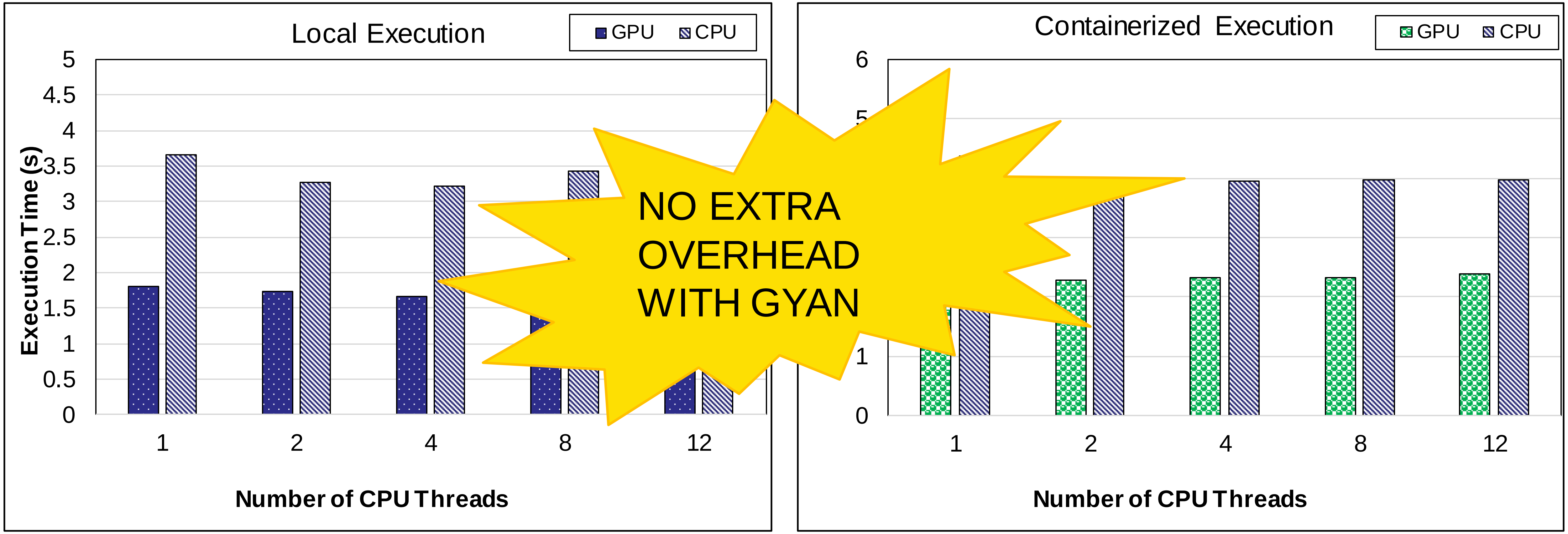


Evaluation Tools: Bonito and Racon

- "Basecalling": Converts raw sequencing data into a sequence of individual nucleotides.
- **Bonito** is a PyTorch-based basecaller provided by Oxford Nanopore Technologies
  - Inspired by the usage of convolutional neural networks (CNNs) in speech recognition.
- **Racon** is a consensus module for raw de novo DNA assembly of long uncorrected reads
  - Consensus generation: An order of magnitude faster than state-of-the-art methods.

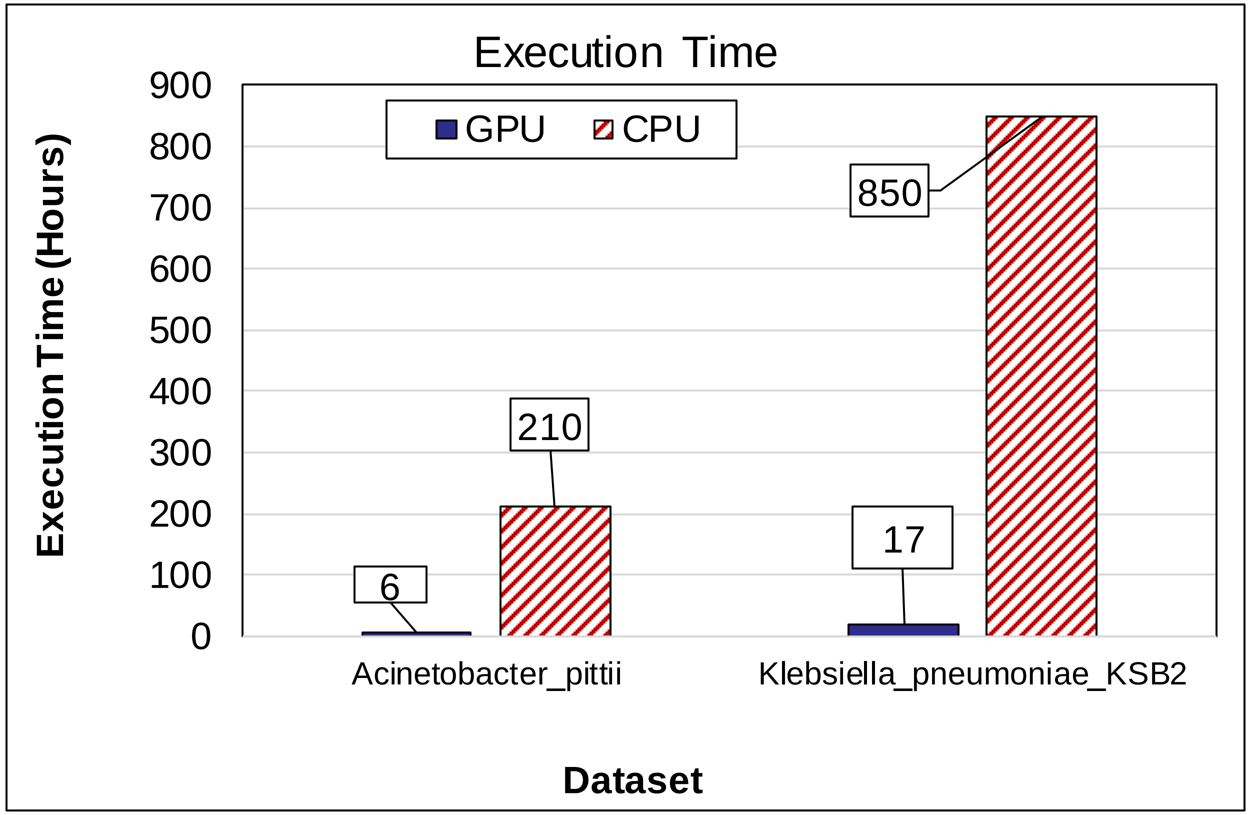
Source: https://gencore.bio.nyu.edu/

Experimental Results for Racon



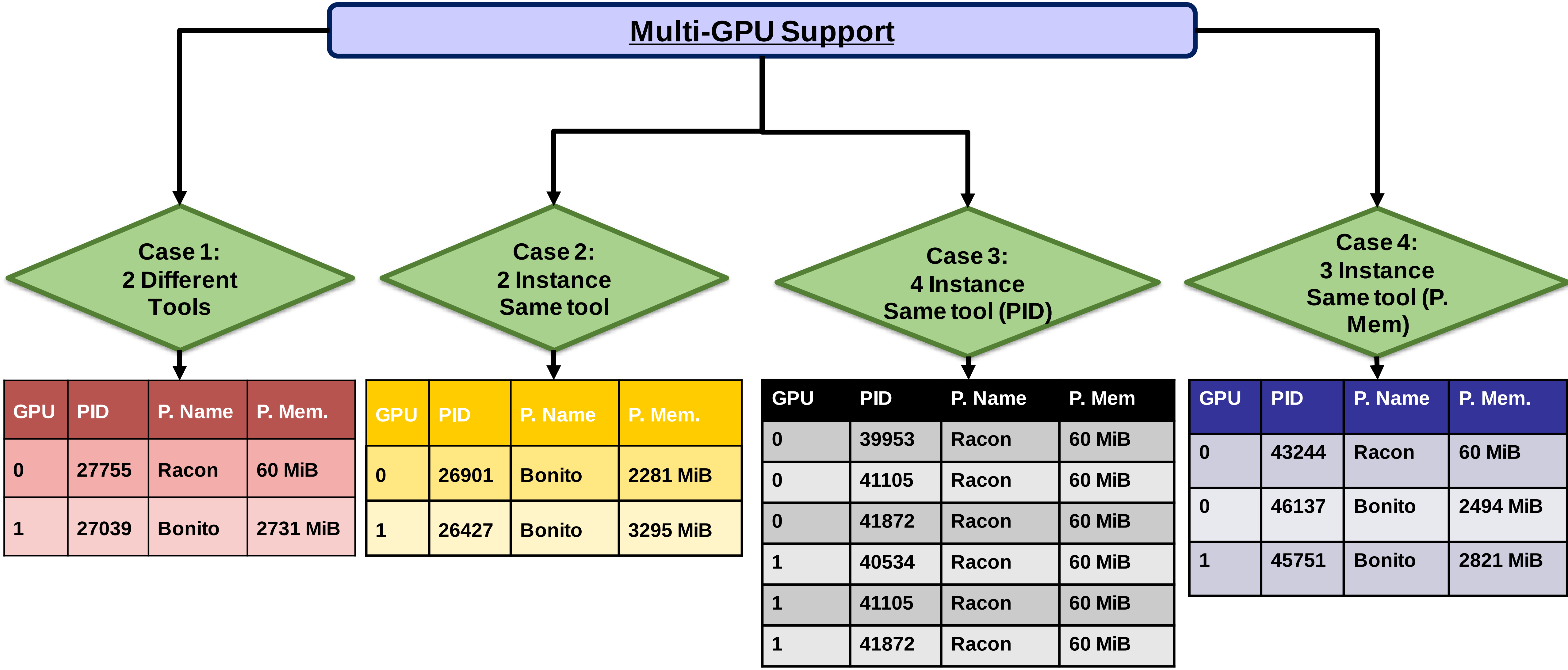
We obtained ~2x speedup using the GPU-supported Racon tool.

Experimental Results for Bonito



We obtained 50x speedup with the both datasets using GPU version of Bonito

Multi-GPU Support



GPU	PID	P. Name	P. Mem.
0	27755	Racon	60 MiB
1	27039	Bonito	2731 MiB

GPU	PID	P. Name	P. Mem.
0	26901	Bonito	2281 MiB
1	26427	Bonito	3295 MiB

GPU	PID	P. Name	P. Mem
0	39953	Racon	60 MiB
0	41105	Racon	60 MiB
0	41872	Racon	60 MiB
1	40534	Racon	60 MiB
1	41105	Racon	60 MiB
1	41872	Racon	60 MiB

GPU	PID	P. Name	P. Mem.
0	43244	Racon	60 MiB
0	46137	Bonito	2494 MiB
1	45751	Bonito	2821 MiB

Overview

- Introduction
- Motivation
- Design of GYAN
- Implementation DetailsEvaluation
- Conclusion

Conclusion

- GYAN: An enhanced version of Galaxy with GPU-support
- Intelligent GPU-aware computation mapping and orchestration support to Galaxy, for researchers to execute the tools in both CPU (or) GPU based on the tool requirements.
- Racon: 2X speedup and Bonito: 50X speedup with no overhead of GYAN
- GYAN's source code will be open-source and it will be merged to public Galaxy's repository.