AMALTHEA-based GPU Response Time Analysis for NVidia's Jetson TX2

Daniel Paredes

Agenda:

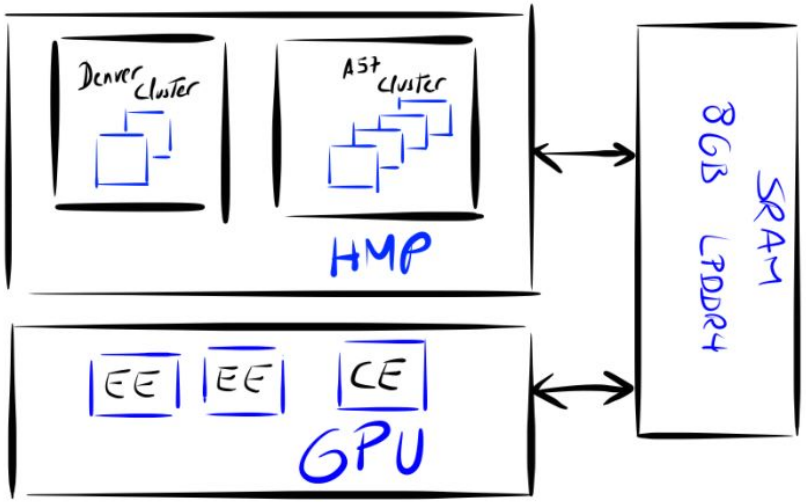
1. Motivation
2. Nvidia GPU Model
3. AMALTHEA Model
4. Response Time Analysis for Jetson TX2's GPU
5. Results
6. Conclusions

Motivation

Motivation

WATERS Industrial Challenge:

- Heterogeneous systems
- Centralized end-to-end architectures.
- Chosen platform: NVIDIA Jetson™ TX2



NVidia GPU Model

Nvidia GPU Model

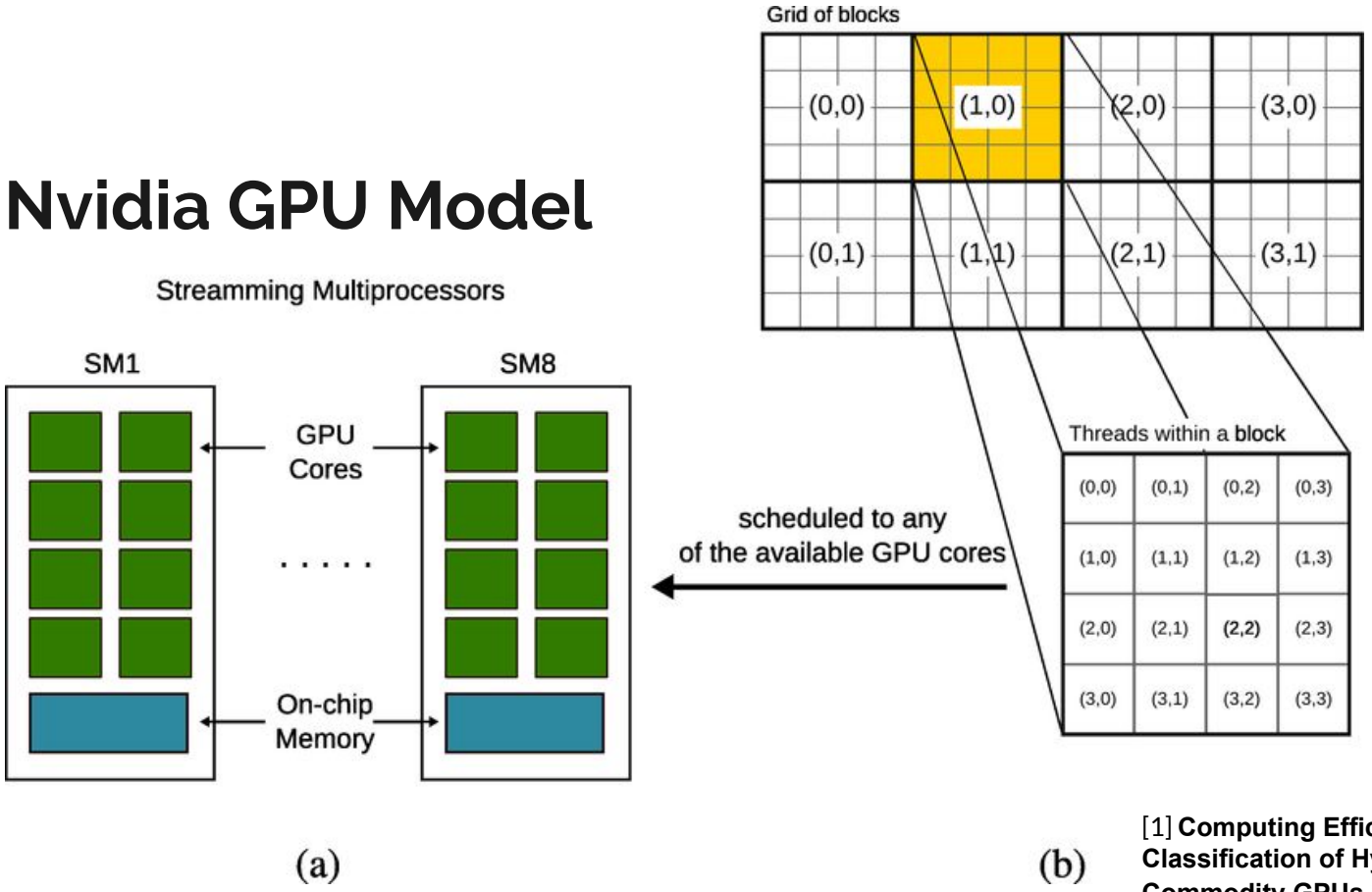
Software Model:

- Threads
- Blocks
- Grids

Hardware Model:

- Streaming Multiprocessors
- Memory

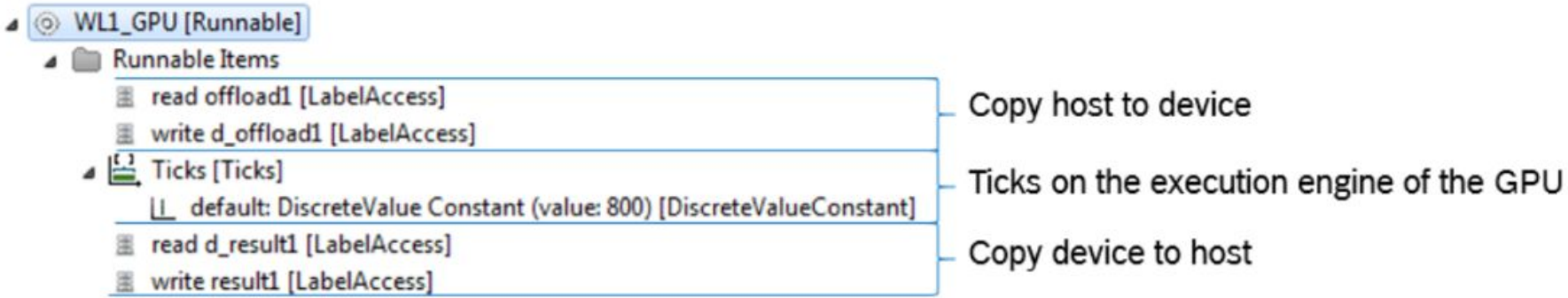
Nvidia GPU Model



[1] **Computing Efficiently Spectral-Spatial Classification of Hyperspectral Images on Commodity GPUs -** Pablo Quesada-Barriuso

AMALTHEA Model

AMALTHEA Model

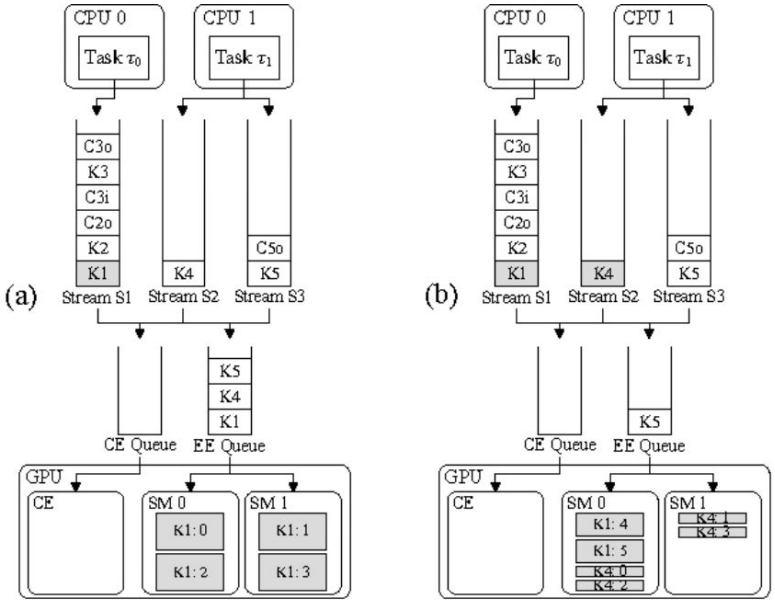


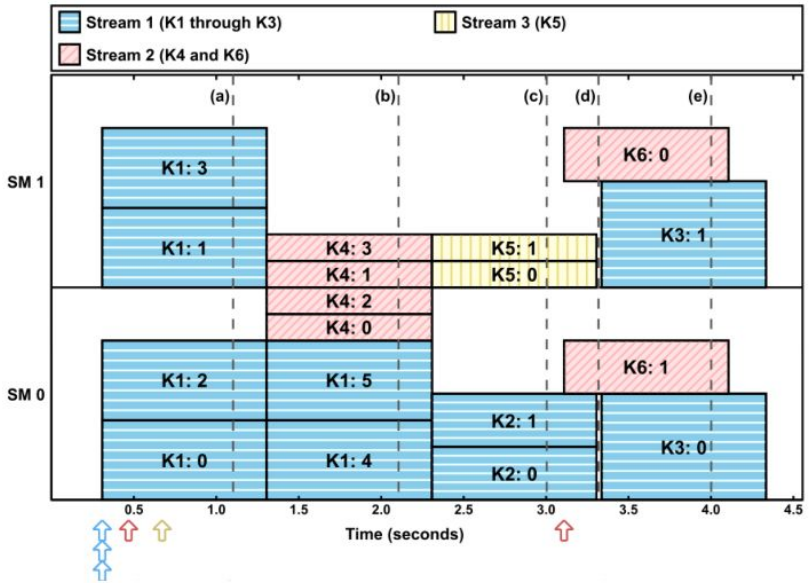
Response Time Analysis

Jetson TX2's GPU

- 2 Streaming Multiprocessors
- FIFO Queues
- Max. threads per Block: 1024
- Max. threads per SM: 2048

GPU Scheduler





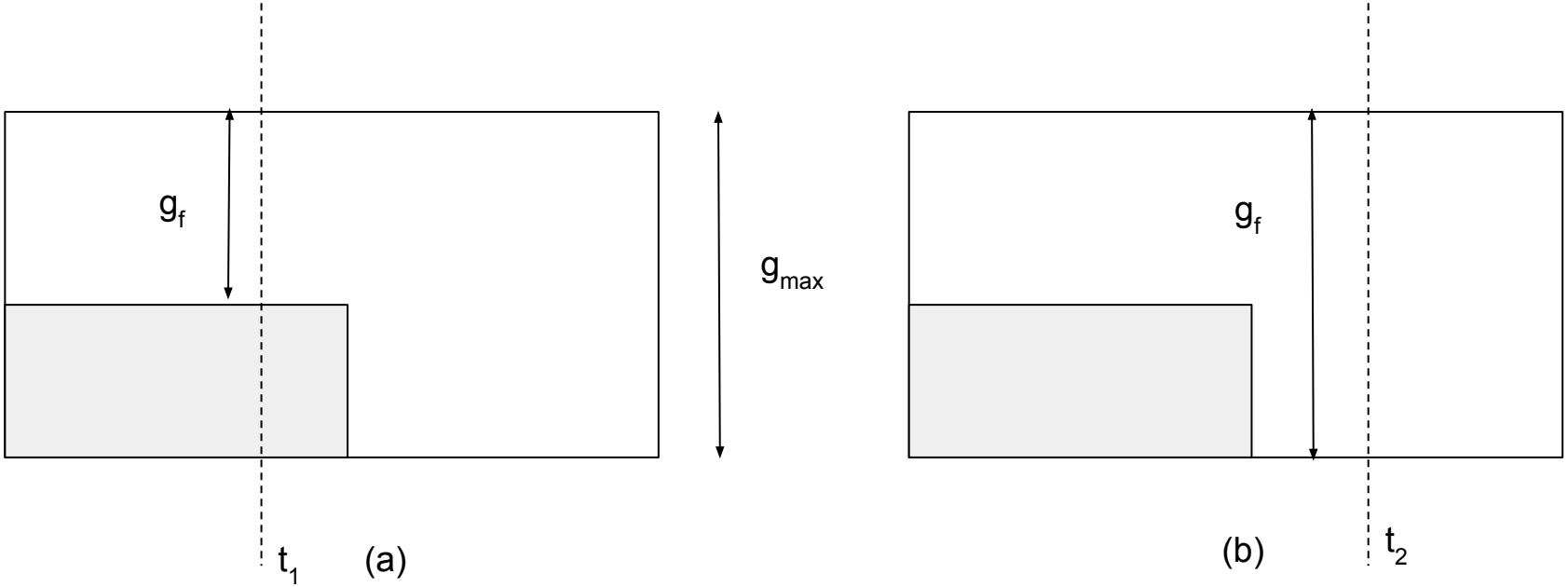
[2] GPU *Scheduling* on the NVIDIA *TX2*: *Hidden* Details Revealed - Tanya Amert et al.

Assumptions

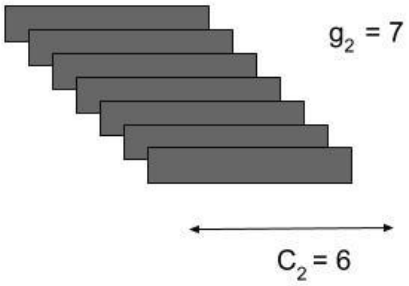
- All blocks have the same amount of threads.
  - Best practices on CUDA programming

- One big Streaming Multiprocessor.
  - #Threads per SM / #threads per block = N

Response Time Calculation

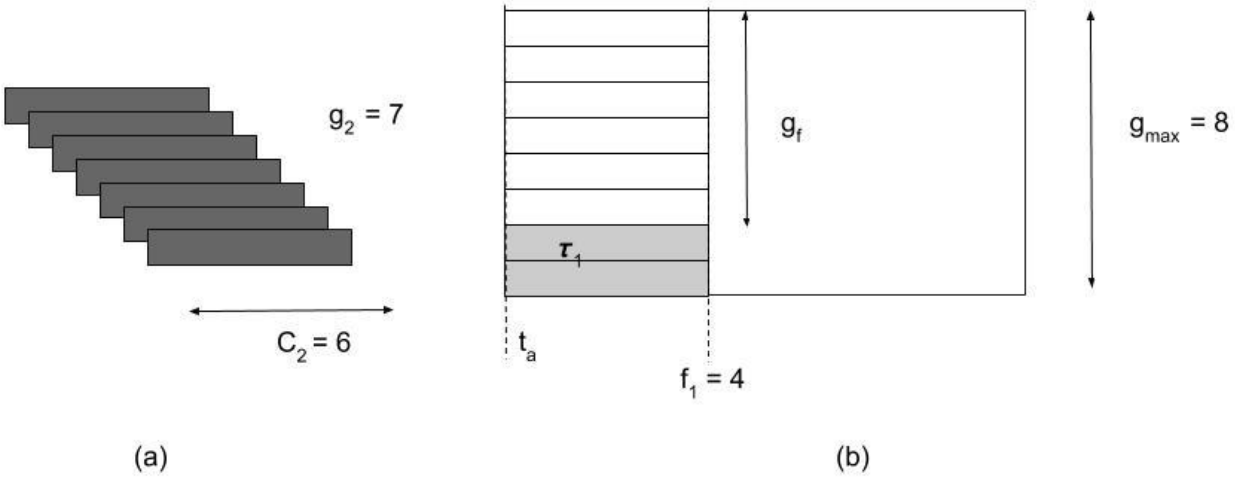


Response Time Calculation

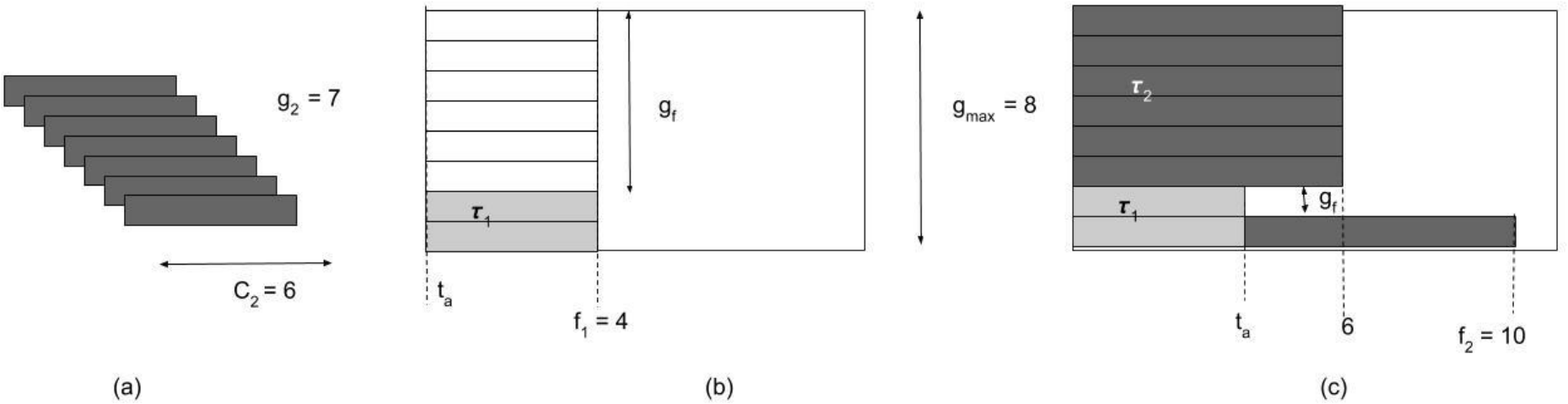


(a)

Response Time Calculation



Response Time Calculation

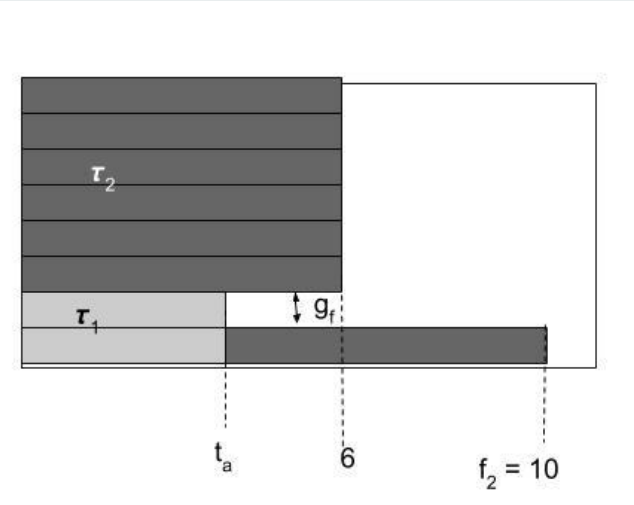


Algorithm

τ : set of kernels
f_i : completion time of kernel i
C_i : thread execution time

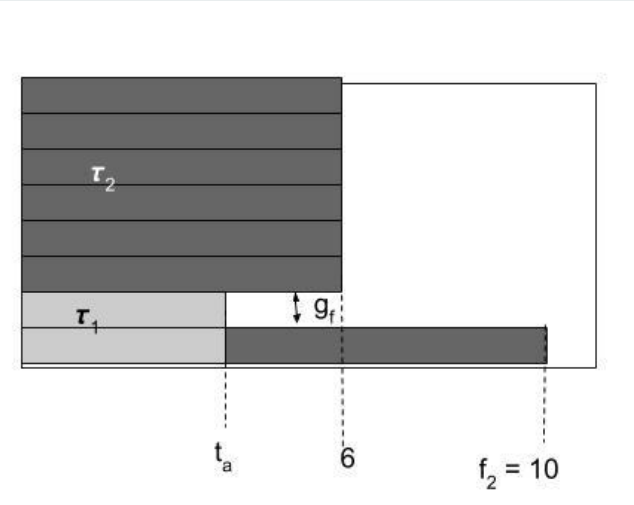
Input : τ
Output: $f_1, \cdots, f_n$
Initialization: $t_a = 0$, $g_f = g_{max}$, $i = 1$
while $i \leq n$ **do**
  if $g_f \geq g_i$ **then**
    $f_i = t_a + C_i$;
    Update g_f and t_a;
    i++ ; // Next kernel
  else
    $g_i = g_i - g_f$;
    Update g_f and t_a;
  end
end

Algorithm



Input : τ
Output: $f_1, \cdots, f_n$
Initialization: $t_a = 0$, $g_f = g_{max}$, $i = 1$
while $i \leq n$ **do**
  if $g_f \geq g_i$ **then**
    $f_i = t_a + C_i$;
    Update g_f and t_a;
    i++ ; // Next kernel
  else
    $g_i = g_i - g_f$;
    Update g_f and t_a;
  end
end

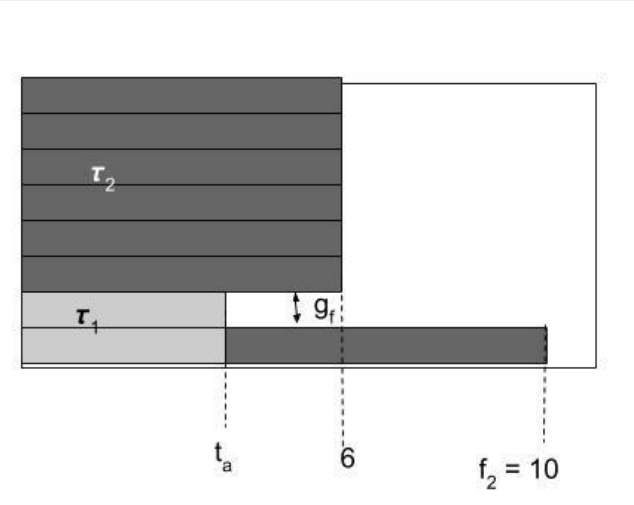
Algorithm



Input : τ
Output: $f_1, \cdots, f_n$

Initialization: $t_a = 0$, $g_f = g_{max}$, $i = 1$, $h = \{\}$
while $i \leq n$ **do**
  if $g_f \geq g_i$ **then**
    $f_i = t_a + C_i$;
    $h = \{h; (f_i, g_i)\}$;
    $t_a = t_a$;
    $g_f = g_f - g_i$;
    i++ ; // Next kernel
  else
    $g_i = g_i - g_f$;
    $h = \{h; (t_a + C_i, g_f)\}$;
    $[t_a, \text{index}] = \min(h[:, 1])$;
    $g_f = h[\text{index}, 2]$;
    Update h;
  end
end

Algorithm



Input : τ
Output: $f_1, \cdots, f_n$

Initialization: $t_a = 0$, $g_f = g_{max}$, $i = 1$, $h = \{\}$
while $i \leq n$ **do**
  if $g_f \geq g_i$ **then**
    $f_i = t_a + C_i$;
    $h = \{h; (f_i, g_i)\}$;
    $t_a = t_a$;
    $g_f = g_f - g_i$;
    i++ ; // Next kernel
  else
    $g_i = g_i - g_f$;
    $h = \{h; (t_a + C_i, g_f)\}$;
    $[t_a, \text{index}] = \min(h[:, 1])$;
    $g_f = h[\text{index}, 2]$;
    Update h;
  end
end

AMALTHEA implementation

```
while i ≤ n do
    if g_f ≥ g_i then
        f_i = t_a + C_i;
        h = {h; (f_i, g_i)};
        t_a = t_a;
        g_f = g_f − g_i;
        i++ ; // Next kernel
    else
        g_i = g_i − g_f;
        h = {h; (t_a + C_i, g_f)};
        [t_a, index] = min(h[:, 1]);
        g_f = h[index, 2];
        Update h;
    end
end
```

```
// Main loop
while ( current_kernel < rList.size() ) {
    if (g_f >= g_i.get(current_kernel) ){
        f.add(current_kernel, t_a + c_i.get(current_kernel) ) ;

        h = updateH(h, f.get(current_kernel), g_i.get(current_kernel) );

        g_f = g_f − g_i.get(current_kernel);
        current_kernel++;
    }
    else {
        g_i.set(current_kernel, g_i.get(current_kernel) − g_f);

        h = updateH(h, t_a + c_i.get(current_kernel), g_f );
        minimumRegisteredTicks = findIndexOfMinValue(h);

        g_f = h.get(minimumRegisteredTicks);
        t_a = minimumRegisteredTicks;

        h.remove(minimumRegisteredTicks);
    }
}
```

AMALTHEA implementation

```
while i ≤ n do
    if g_f ≥ g_i then
        f_i = t_a + C_i;
        h = {h; (f_i, g_i)};
        t_a = t_a;
        g_f = g_f − g_i;
        i++ ; // Next kernel
    else
        g_i = g_i − g_f;
        h = {h; (t_a + C_i, g_f)};
        [t_a, index] = min(h[:, 1]);
        g_f = h[index, 2];
        Update h;
    end
end
```

```
// Main loop
while ( current_kernel < rList.size() ) {
    if (g_f >= g_i.get(current_kernel) ){
        f.add(current_kernel, t_a + c_i.get(current_kernel) ) ;

        h = updateH(h, f.get(current_kernel), g_i.get(current_kernel) );

        g_f = g_f − g_i.get(current_kernel);
        current_kernel++;
    }
    else {
        g_i.set(current_kernel, g_i.get(current_kernel) − g_f);

        h = updateH(h, t_a + c_i.get(current_kernel), g_f );
        minimumRegisteredTicks = findIndexOfMinValue(h);

        g_f = h.get(minimumRegisteredTicks);
        t_a = minimumRegisteredTicks;

        h.remove(minimumRegisteredTicks);
    }
}
```

AMALTHEA implementation

```
while i ≤ n do
    if g_f ≥ g_i then
        f_i = t_a + C_i;
        h = {h; (f_i, g_i)};
        t_a = t_a;
        g_f = g_f − g_i;
        i++ ; // Next kernel
    else
        g_i = g_i − g_f;
        h = {h; (t_a + C_i, g_f)};
        [t_a, index] = min(h[:, 1]);
        g_f = h[index, 2];
        Update h;
    end
end
```

```
// Main loop
while ( current_kernel < rList.size() ) {
    if (g_f >= g_i.get(current_kernel) ){
        f.add(current_kernel, t_a + c_i.get(current_kernel) ) ;

        h = updateH(h, f.get(current_kernel), g_i.get(current_kernel) );

        g_f = g_f − g_i.get(current_kernel);
        current_kernel++;
    }
    else {
        g_i.set(current_kernel, g_i.get(current_kernel) − g_f);

        h = updateH(h, t_a + c_i.get(current_kernel), g_f );
        minimumRegisteredTicks = findIndexOfMinValue(h);

        g_f = h.get(minimumRegisteredTicks);
        t_a = minimumRegisteredTicks;

        h.remove(minimumRegisteredTicks);
    }
}
```

AMALTHEA implementation

```
while i ≤ n do
    if g_f ≥ g_i then
        f_i = t_a + C_i;
        h = {h; (f_i, g_i)};
        t_a = t_a;
        g_f = g_f − g_i;
        i++ ; // Next kernel
    else
        g_i = g_i − g_f;
        h = {h; (t_a + C_i, g_f)};
        [t_a, index] = min(h[:, 1]);
        g_f = h[index, 2];
        Update h;
    end
end
```

```
// Main loop
while ( current_kernel < rList.size() ) {
    if (g_f >= g_i.get(current_kernel) ){
        f.add(current_kernel, t_a + c_i.get(current_kernel) ) ;

        h = updateH(h, f.get(current_kernel), g_i.get(current_kernel) );

        g_f = g_f − g_i.get(current_kernel);
        current_kernel++;
    }
    else {
        g_i.set(current_kernel, g_i.get(current_kernel) − g_f);

        h = updateH(h, t_a + c_i.get(current_kernel), g_f );
        minimumRegisteredTicks = findIndexOfMinValue(h);

        g_f = h.get(minimumRegisteredTicks);
        t_a = minimumRegisteredTicks;

        h.remove(minimumRegisteredTicks);
    }
}
```

AMALTHEA implementation

```
while i ≤ n do
    if g_f ≥ g_i then
        f_i = t_a + C_i;
        h = {h; (f_i, g_i)};
        t_a = t_a;
        g_f = g_f − g_i;
        i++ ; // Next kernel
    else
        g_i = g_i − g_f;
        h = {h; (t_a + C_i, g_f)};
        [t_a, index] = min(h[:, 1]);
        g_f = h[index, 2];
        Update h;
    end
end
```

```
// Main loop
while ( current_kernel < rList.size() ) {
    if (g_f >= g_i.get(current_kernel) ){
        f.add(current_kernel, t_a + c_i.get(current_kernel) ) ;

        h = updateH(h, f.get(current_kernel), g_i.get(current_kernel) );

        g_f = g_f − g_i.get(current_kernel);
        current_kernel++;
    }
    else {
        g_i.set(current_kernel, g_i.get(current_kernel) − g_f);

        h = updateH(h, t_a + c_i.get(current_kernel), g_f );
        minimumRegisteredTicks = findIndexOfMinValue(h);

        g_f = h.get(minimumRegisteredTicks);
        t_a = minimumRegisteredTicks;

        h.remove(minimumRegisteredTicks);
    }
}
```

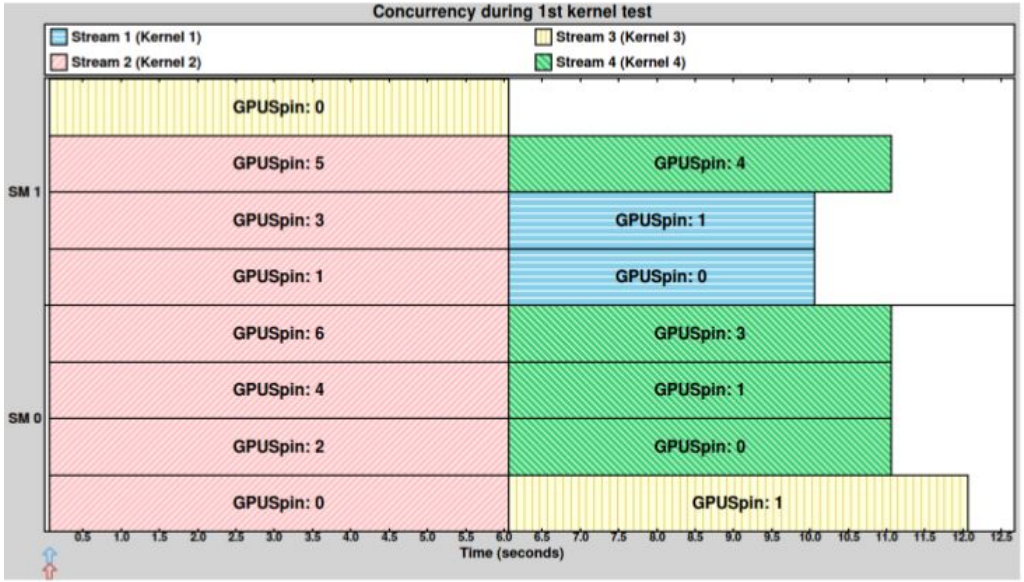
Results

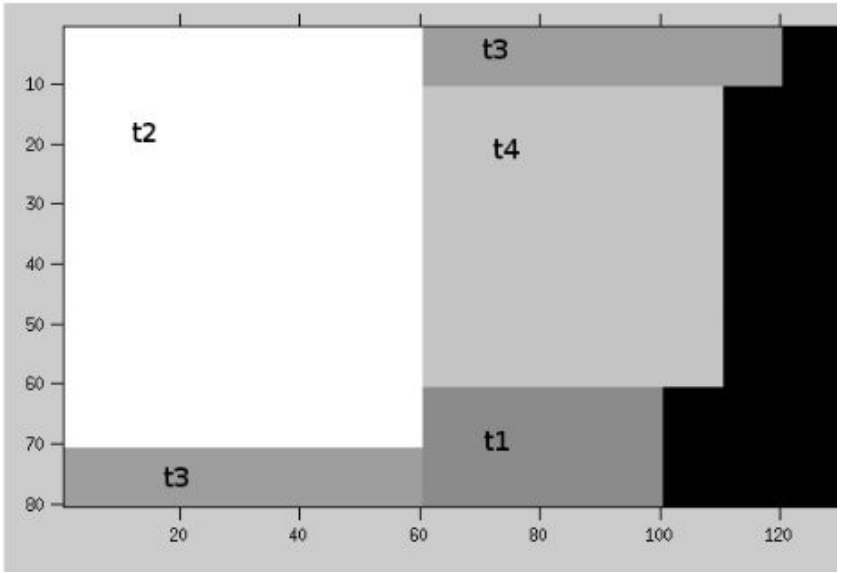
Set-up

- Dummy kernels on Jetson TX2
- AMALTHEA implementation

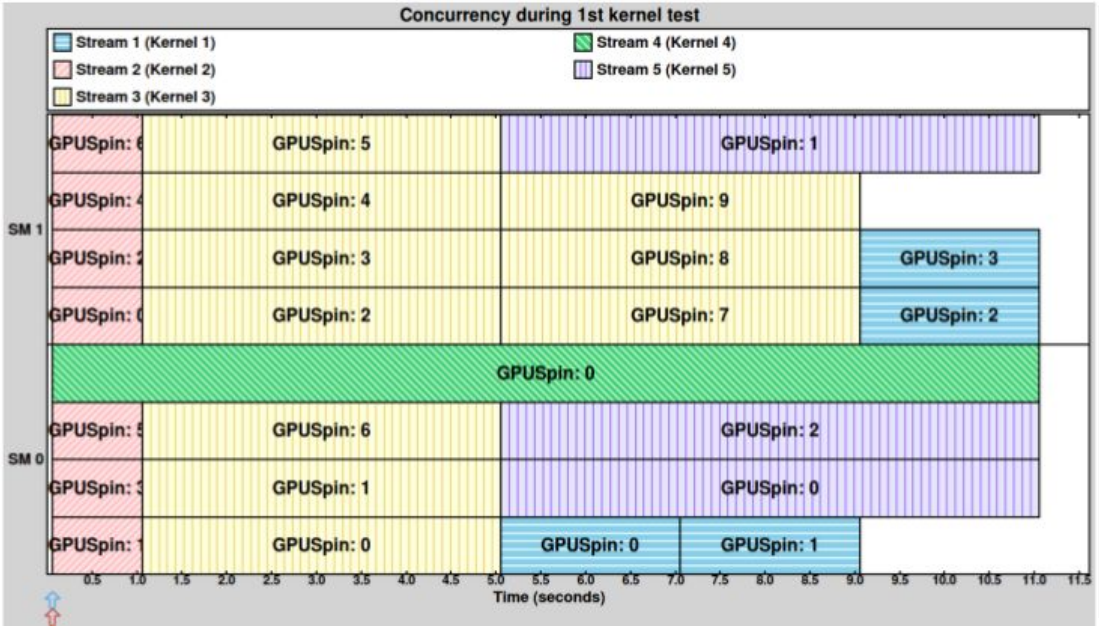
  - Timing measurement
  - Block allocation diagrams

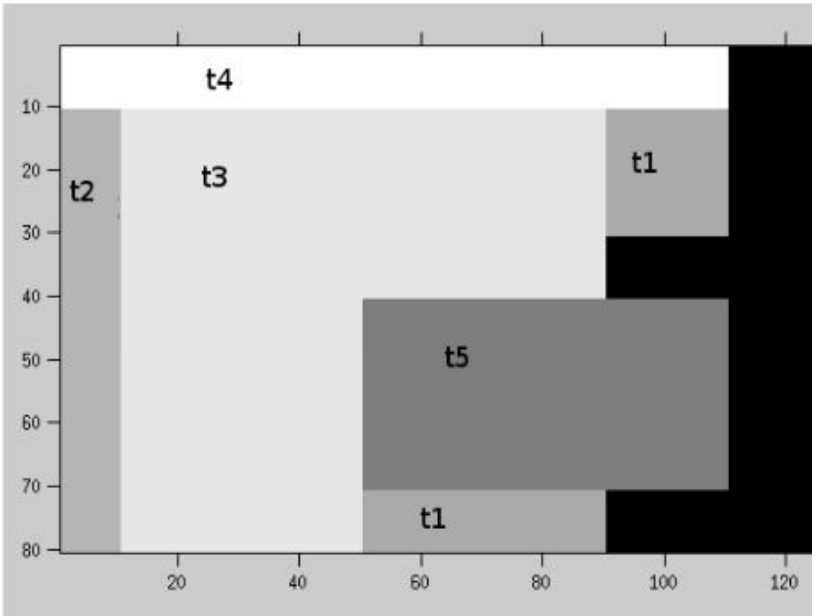
Results


Concurrency during 1st kernel test
Stream 1 (Kernel 1)
Stream 2 (Kernel 2)
Stream 3 (Kernel 3)
Stream 4 (Kernel 4)
SM 1
SM 0
GPUSpin: 0
GPUSpin: 5
GPUSpin: 4
GPUSpin: 3
GPUSpin: 1
GPUSpin: 0
GPUSpin: 6
GPUSpin: 3
GPUSpin: 4
GPUSpin: 1
GPUSpin: 2
GPUSpin: 0
GPUSpin: 0
GPUSpin: 1
0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0 8.5 9.0 9.5 10.0 10.5 11.0 11.5 12.0 12.5
Time (seconds)

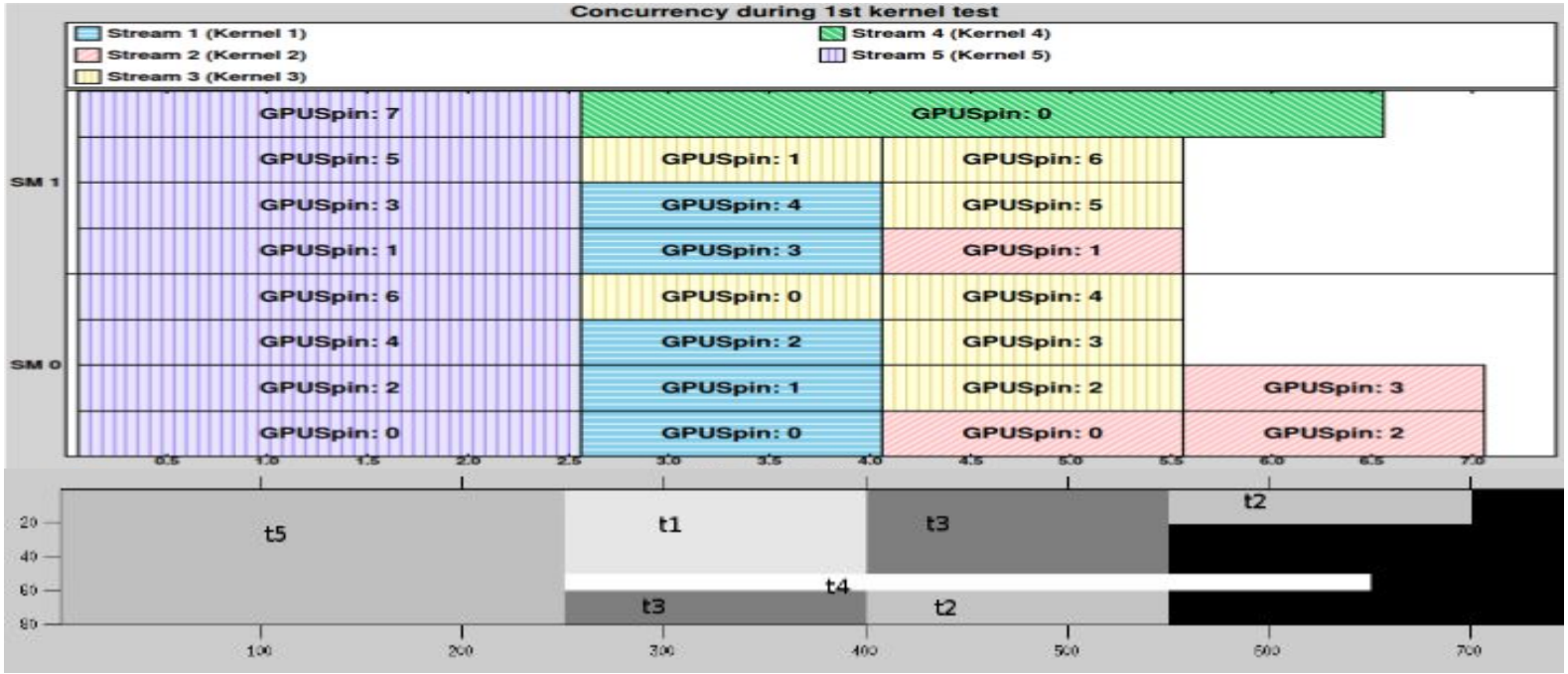

t3
t2
t4
t1
t3
10
20
30
40
50
60
70
80
20
40
60
80
100
120

Results


Concurrency during 1st kernel test
Stream 1 (Kernel 1)
Stream 2 (Kernel 2)
Stream 3 (Kernel 3)
Stream 4 (Kernel 4)
Stream 5 (Kernel 5)
SM 1
SM 0
GPUSpin: 5
GPUSpin: 1
GPUSpin: 4
GPUSpin: 9
GPUSpin: 3
GPUSpin: 8
GPUSpin: 2
GPUSpin: 7
GPUSpin: 0
GPUSpin: 6
GPUSpin: 2
GPUSpin: 1
GPUSpin: 0
GPUSpin: 0
GPUSpin: 1
Time (seconds)


t4
t2
t3
t1
t5
t1
10
20
30
40
50
60
70
80
20
40
60
80
100
120

Results


Concurrency during 1st kernel test
Stream 1 (Kernel 1)
Stream 2 (Kernel 2)
Stream 3 (Kernel 3)
Stream 4 (Kernel 4)
Stream 5 (Kernel 5)
SM 1
SM 0
GPUSpin: 7
GPUSpin: 0
GPUSpin: 5
GPUSpin: 1
GPUSpin: 6
GPUSpin: 3
GPUSpin: 4
GPUSpin: 5
GPUSpin: 1
GPUSpin: 3
GPUSpin: 1
GPUSpin: 6
GPUSpin: 0
GPUSpin: 4
GPUSpin: 4
GPUSpin: 2
GPUSpin: 3
GPUSpin: 2
GPUSpin: 1
GPUSpin: 2
GPUSpin: 3
GPUSpin: 0
GPUSpin: 0
GPUSpin: 0
GPUSpin: 2
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
t5
t1
t3
t2
t4
t3
t2
20
40
60
80
100
200
300
400
500
600
700

Conclusions and Future work

Conclusions

- Simple and easy to implement algorithm
- Accuracy in calculation completion times
- Integration with AMALTHEA models
- Simulation of complex use cases on AMALTHEA models

Future Work

- Automatic CUDA code generation
- Consideration of memory transactions
- Not fully understand of Jetson TX2’s GPU behaviour

References:

1. Quesada-Barriuso, Pablo & Argüello, Francisco & B. Heras, Dora. (2013). Computing Efficiently Spectral-Spatial Classification of Hyperspectral Images on Commodity GPUs. 10.1007/978-3-319-01649-8_2.
2. T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, “GPU Scheduling on the Nvidia TX2: Hidden Details Revealed,” in 2017 ieee real-time systems symposium (rtss), 2017, pp. 104–115.

AMALTHEA-based GPU Response Time Analysis for NVidia's Jetson TX2

Daniel Paredes