

## Proyecto 3

## Enlaces

- Repositorio

## Integrantes

- **Guía conceptual - Brandon Reyes**
- **Analista - Carlos Valladares**
- **Programador principal - Josué Say**

## Fase 1

1. Define cuántas galaxias tendrá tu universo (bloques) y cuántas estrellas hay por galaxia (hilos).
  - **Por defecto 4 galaxiaas con 3 estrellas**
2. Cada estrella debe calcular su propio brillo (un número entre 0 y 9) y mostrar en consola algo similar a:
  - Galaxia 0 - Estrella 0 → Brillo: 7
  - Galaxia 0 - Estrella 1 → Brillo: 2
  - Galaxia 1 - Estrella 0 → Brillo: 4
  - Galaxia 1 - Estrella 1 → Brillo: 9

The screenshot displays the Visual Studio Code editor interface. The top menu bar includes 'Archivo', 'Editar', 'Selección', 'Ver', 'Ejecutar', and 'Ayuda'. The top toolbar contains icons for file operations and execution. The left sidebar shows the 'EXPLORADOR' (Explorer) view with a file tree containing folders like 'imagenes' and files like 'fase1.png', 'fase2.png', 'f1', 'f2', 'fase1.cu', 'fase2.cu', 'guia.pdf', and 'informe.md'. The main editor area shows the 'fase1.cu' file with the following C++ code:

```

1 //informe.md
2
3 #include <cuda_runtime.h>
4 #include <device_launch_parameters.h>
5
6 //fase1.cu
7
8 //fase2.cu
9
10 //f1
11 //f2
12 //f3
13 //f4
14 //f5
15 //f6
16 //f7
17 //f8
18 //f9
19 //f10
20 //f11
21 //f12
22 //f13
23 //f14
24 //f15
25 //f16
26 //f17
27 //f18
28 //f19
29 //f20
30 //f21
31 //f22
32 //f23
33 //f24
34 //f25
35 //f26
36 //f27
37 //f28
38 //f29
39 //f30
40 //f31
41 //f32
42 //f33
43 //f34
44 //f35
45 //f36
46 //f37
47 //f38
48 //f39
49 //f40
50 //f41
51 //f42
52 //f43
53 //f44
54 //f45
55 //f46
56 //f47
57 //f48
58 //f49
59 //f50
60 //f51
61 //f52
62 //f53
63 //f54
64 //f55
65 //f56
66 //f57
67 //f58
68 //f59
69 //f60
70 //f61
71 //f62
72 //f63
73 //f64
74 //f65
75 //f66
76 //f67
77 //f68
78 //f69
79 //f70
80 //f71
81 //f72
82 //f73
83 //f74
84 //f75
85 //f76
86 //f77
87 //f78
88 //f79
89 //f80
90 //f81
91 //f82
92 //f83
93 //f84
94 //f85
95 //f86
96 //f87
97 //f88
98 //f89
99 //f90
100 //f91
101 //f92
102 //f93
103 //f94
104 //f95
105 //f96
106 //f97
107 //f98
108 //f99
109 //f100
110 //f101
111 //f102
112 //f103
113 //f104
114 //f105
115 //f106
116 //f107
117 //f108
118 //f109
119 //f110
120 //f111
121 //f112
122 //f113
123 //f114
124 //f115
125 //f116
126 //f117
127 //f118
128 //f119
129 //f120
130 //f121
131 //f122
132 //f123
133 //f124
134 //f125
135 //f126
136 //f127
137 //f128
138 //f129
139 //f130
140 //f131
141 //f132
142 //f133
143 //f134
144 //f135
145 //f136
146 //f137
147 //f138
148 //f139
149 //f140
150 //f141
151 //f142
152 //f143
153 //f144
154 //f145
155 //f146
156 //f147
157 //f148
158 //f149
159 //f150
160 //f151
161 //f152
162 //f153
163 //f154
164 //f155
165 //f156
166 //f157
167 //f158
168 //f159
169 //f160
170 //f161
171 //f162
172 //f163
173 //f164
174 //f165
175 //f166
176 //f167
177 //f168
178 //f169
179 //f170
180 //f171
181 //f172
182 //f173
183 //f174
184 //f175
185 //f176
186 //f177
187 //f178
188 //f179
189 //f180
190 //f181
191 //f182
192 //f183
193 //f184
194 //f185
195 //f186
196 //f187
197 //f188
198 //f189
199 //f190
200 //f191
201 //f192
202 //f193
203 //f194
204 //f195
205 //f196
206 //f197
207 //f198
208 //f199
209 //f200
210 //f201
211 //f202
212 //f203
213 //f204
214 //f205
215 //f206
216 //f207
217 //f208
218 //f209
219 //f210
220 //f211
221 //f212
222 //f213
223 //f214
224 //f215
225 //f216
226 //f217
227 //f218
228 //f219
229 //f220
230 //f221
231 //f222
232 //f223
233 //f224
234 //f225
235 //f226
236 //f227
237 //f228
238 //f229
239 //f230
240 //f231
241 //f232
242 //f233
243 //f234
244 //f235
245 //f236
246 //f237
247 //f238
248 //f239
249 //f240
250 //f241
251 //f242
252 //f243
253 //f244
254 //f245
255 //f246
256 //f247
257 //f248
258 //f249
259 //f250
260 //f251
261 //f252
262 //f253
263 //f254
264 //f255
265 //f256
266 //f257
267 //f258
268 //f259
269 //f260
270 //f261
271 //f262
272 //f263
273 //f264
274 //f265
275 //f266
276 //f267
277 //f268
278 //f269
279 //f270
280 //f271
281 //f272
282 //f273
283 //f274
284 //f275
285 //f276
286 //f277
287 //f278
288 //f279
289 //f280
290 //f281
291 //f282
292 //f283
293 //f284
294 //f285
295 //f286
296 //f287
297 //f288
298 //f289
299 //f290
300 //f291
301 //f292
302 //f293
303 //f294
304 //f295
305 //f296
306 //f297
307 //f298
308 //f299
309 //f300
310 //f301
311 //f302
312 //f303
313 //f304
314 //f305
315 //f306
316 //f307
317 //f308
318 //f309
319 //f310
320 //f311
321 //f312
322 //f313
323 //f314
324 //f315
325 //f316
326 //f317
327 //f318
328 //f319
329 //f320
330 //f321
331 //f322
332 //f323
333 //f324
334 //f325
335 //f326
336 //f327
337 //f328
338 //f329
339 //f330
340 //f331
341 //f332
342 //f333
343 //f334
344 //f335
345 //f336
346 //f337
347 //f338
348 //f339
349 //f340
350 //f341
351 //f342
352 //f343
353 //f344
354 //f345
355 //f346
356 //f347
357 //f348
358 //f349
359 //f350
360 //f351
361 //f352
362 //f353
363 //f354
364 //f355
365 //f356
366 //f357
367 //f358
368 //f359
369 //f360
370 //f361
371 //f362
372 //f363
373 //f364
374 //f365
375 //f366
376 //f367
377 //f368
378 //f369
379 //f370
380 //f371
381 //f372
382 //f373
383 //f374
384 //f375
385 //f376
386 //f377
387 //f378
388 //f379
389 //f380
390 //f381
391 //f382
392 //f383
393 //f384
394 //f385
395 //f386
396 //f387
397 //f388
398 //f389
399 //f390
400 //f391
401 //f392
402 //f393
403 //f394
404 //f395
405 //f396
406 //f397
407 //f398
408 //f399
409 //f400
410 //f401
411 //f402
412 //f403
413 //f404
414 //f405
415 //f406
416 //f407
417 //f408
418 //f409
419 //f410
420 //f411
421 //f412
422 //f413
423 //f414
424 //f415
425 //f416
426 //f417
427 //f418
428 //f419
429 //f420
430 //f421
431 //f422
432 //f423
433 //f424
434 //f425
435 //f426
436 //f427
437 //f428
438 //f429
439 //f430
440 //f431
441 //f432
442 //f433
443 //f434
444 //f435
445 //f436
446 //f437
447 //f438
448 //f439
449 //f440
450 //f441
451 //f442
452 //f443
453 //f444
454 //f4
```

3. Analiza visualmente el orden de impresión: ¿es secuencial o aleatorio?
- El orden de impresión parece ser aleatorio porque no aparece en el orden de galaxias  $0 \rightarrow 4$ , aparecen aleatoriamente.

## Preguntas de validación

### 1. ¿Qué relación existe entre `blockIdx.x` y el número de galaxia impreso?

`blockIdx` es directamente el número de galaxias, donde cada bloque representa una galaxia y el índice `x` corresponde al número que se imprime, por ejemplo si colocamos 4 galaxias, CUDA crea bloques del 0 al 3.

### 2. ¿Qué representa `threadIdx.x` dentro de tu simulación?

`threadIdx.x` representa el número de estrellas dentro de cada galaxia, los hilos comparten la misma galaxia (`blockIdx.x`) pero tienen distinto índice de estrella.

### 3. Si duplicas la cantidad de estrellas, ¿cambia el orden de impresión o solo el tamaño de la salida?

Al duplicar la cantidad de estrellas únicamente cambia el tamaño de salida, pero el orden de impresión real sigue siendo el mismo porque el comportamiento sigue siendo el mismo.

### 4. ¿Qué representa el “brillo” dentro de este modelo paralelo?

El brillo es un cálculo que hace cada hilo de manera independiente, el brillo prácticamente simboliza el output independiente que cada hilo hace al ejecutar la misma función pero con distintos índices.

## Fase 2

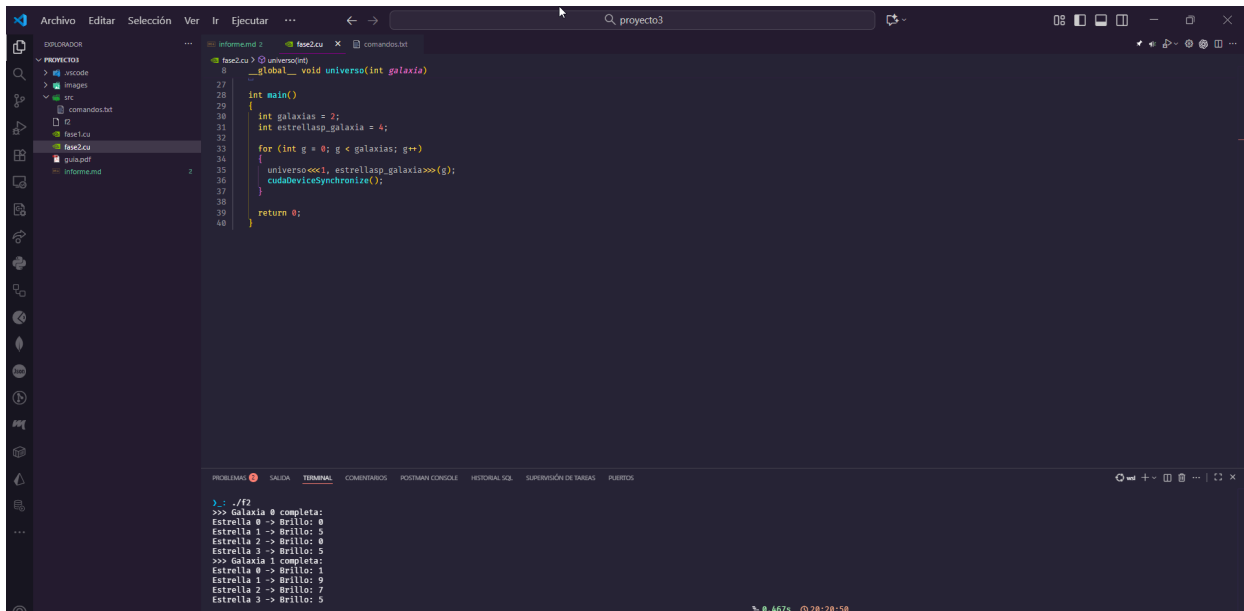
### Instrucciones

1. Introduce un mecanismo de sincronización para que las estrellas esperen entre sí antes de imprimir.
2. Logra que cada galaxia imprima sus estrellas juntas, como si todas brillaran al mismo tiempo.
3. Resultado esperado:

Resultado esperado

```
>>> Galaxia 0 completa:
Estrella 0 -> Brillo 4
Estrella 1 -> Brillo 9
Estrella 2 -> Brillo 3
Estrella 3 -> Brillo 1
>>> Galaxia 1 completa:
Estrella 0 -> Brillo 7
```

Estrella 1 -> Brillo 2  
Estrella 2 -> Brillo 5  
Estrella 3 -> Brillo 6



```
1 // __syncthreads()
2
3 global __void universo(int galaxia)
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28 int main()
29 {
30     int galaxias = 2;
31     int estrellas_galaxia = 4;
32
33     for (int g = 0; g < galaxias; g++)
34     {
35         universoect, estrellas_galaxia(g);
36         cudaDeviceSynchronize();
37     }
38     return 0;
39 }
40 }
```

```
>>> Galaxia 0 completa:
Estrella 0 -> Brillo: 0
Estrella 1 -> Brillo: 5
Estrella 2 -> Brillo: 0
Estrella 3 -> Brillo: 5
>>> Galaxia 1 completa:
Estrella 0 -> Brillo: 1
Estrella 1 -> Brillo: 0
Estrella 2 -> Brillo: 7
Estrella 3 -> Brillo: 5
```

## Preguntas de validación

### 1. ¿Qué ocurre si eliminamos la sincronización?

Las estrellas se imprimirán en desorden, intercaladas con las demás galaxias, perderíamos la cohesión del bloque.

### 2. ¿Qué significa que `__syncthreads()` solo sincroniza dentro de un bloque?

Significa que solo coordina estrellas dentro de la misma galaxia pero no puede detener ni coordinar estrellas que pertenecen a otros bloques.

### 3. ¿Por qué la sincronización entre galaxias (bloques diferentes) no es posible directamente?

CUDA no permite sincronizar bloques dentro de un mismo kernel porque cada bloque puede ejecutarse en cualquier SM, en cualquier orden y en diferentes momentos.

### 4. ¿Qué tipo de errores podrían aparecer si las estrellas imprimen sin coordinarse?

- Impresiones mezcladas o en orden aleatorio
- Valores incompletos o escritos demasiado pronto (en el caso de galaxias saldría primero)
- Resultados visualmente caóticos
- Se vuelve imposible interpretar el brillo de cada galaxia

### Fase 3

1. Crea una memoria compartida dentro de cada bloque donde las estrellas almacenen sus brillos.
2. Una vez que todos hayan calculado su valor, coordina que solo una estrella (por ejemplo, la de índice 0) calcule y muestre el promedio general del bloque.
3. Presenta el resultado así:

```
>>> Galaxia 0 - Brillo promedio: 5.6
>>> Galaxia 1 - Brillo promedio: 8.1
>>> Galaxia 2 - Brillo promedio: 3.4
```
4. Muestra cuál galaxia es la más brillante (mayor promedio).

```

1  #include <stdio.h>
2  #include <cuda_runtime.h>
3
4  __device__ int calcularBrillo(int galaxia, int estrella)
5  {
6      return (galaxia * galaxia + estrella * 5 + galaxia * estrella * 3) % 10;
7  }
8
9  __global__ void universo(int galaxia, float *promedios)
10 {
11     int estrella = threadIdx.x;
12     __shared__ int brillos[1024]; // guardar el brillo de estrella
13
14     int brillo = calcularBrillo(galaxia, estrella);
15     brillos[estrella] = brillo;
16
17     __syncthreads(); // sincronizarlos
18
19     if (estrella == 0)
20     {
21         float suma = 0;
22         for (int i = 0; i < blockDim.x; i++)
23         {
24             suma += brillos[i];
25         }
26         float promedio = suma / blockDim.x;
27         promedios[galaxia] = promedio;
28         printf(">>> Galaxia %d - Brillo promedio: %.1f\n", galaxia, promedio);
29     }
30 }
31
32 int main()
33 {
34     int galaxias = 3;
35     int estrellasp_galaxia = 4;
36
37     float *d_promedios;
38     float h_promedios[3];
39
40     cudaMalloc(&d_promedios, galaxias * sizeof(float));
41
42     for (int g = 0; g < galaxias; g++)
43     {
44         universo<<<1, estrellasp_galaxia>>>>(g, d_promedios);
45         cudaDeviceSynchronize();
46     }
47
48     cudaMemcpy(h_promedios, d_promedios, galaxias * sizeof(float), cudaMemcpyDeviceToHost);
49
50     int galaxia_max = 0;
51     float brillo_max = h_promedios[0];
52
53     for (int g = 1; g < galaxias; g++)
54     {
55         if (h_promedios[g] > brillo_max)
56         {
57             brillo_max = h_promedios[g];
58             galaxia_max = g;
59         }
60     }
61
62     printf("Galaxia mas brillante: %d con brillo promedio: %.1f\n", galaxia_max, brillo_max);
63
64     cudaFree(d_promedios);
65
66     return 0;
67 }
68

```

## Preguntas de validación

### 1. ¿Por qué es útil la memoria compartida en este contexto?

La memoria compartida es útil ya que permite de que todas las estrellas (que serían los hilos) dentro de una galaxia (el bloque) trabajen con un espacio común de datos.

### 2. ¿Qué ventaja tiene frente al uso de memoria global?

Son dos grandes ventajas respecto a la memoria global, velocidad muchísimo mayor y menor latencia y mayor coherencia, la memoria compartida hace que el cálculo del promedio sea mucho más rápido, ordenado y eficiente mientras que si lo hacemos en memoria global sería mucho más lento y con más riesgo de incoherencia.

### 3. ¿Qué pasaría si más de una estrella intenta escribir al mismo tiempo en la misma posición?

Si más de una estrella intenta escribir al mismo tiempo en la misma posición los valores se sobrescriben de manera impredecible, además que el resultado final depende del hilo que escribió más rápido y se genera un comportamiento no determinista y difícil de depurar, entonces el promedio final sería incorrecto o incoherente. Prácticamente si dos hilos están escribiendo en la misma posición sin coordinación sería igual a un dato corrupto.

### 4. ¿Qué refleja el promedio del brillo respecto al comportamiento de la GPU?

Refleja la cooperación de los hilos entre ellos en un mismo bloque, prácticamente el promedio del brillo simboliza cómo la GPU combina trabajo paralelo para reducir un resultado global por bloque.

## Fase 4

### Brandon Reyes

### 1. ¿Qué aprendiste sobre cómo CUDA distribuye el trabajo entre hilos y bloques?

Entendí que CUDA divide el trabajo en bloques y dentro de cada bloque los hilos se encargan de los cálculos. Cada quien hace su parte sin depender de los demás.

### 2. ¿Qué fue lo más difícil de entender del paralelismo?

Me costó aceptar que las cosas no tienen un orden fijo y que todo corre al mismo tiempo. Eso al inicio se siente raro porque uno espera que todo vaya en secuencia.

**3. Si pudieras mejorar el laboratorio, ¿qué cambio harías en el algoritmo?**

Tal vez agregar una parte donde cada galaxia tenga un comportamiento diferente, algo que no sea tan repetitivo.

**4. ¿Qué analogía del mundo real usarías para explicar el concepto de “sincronización de hilos”?**

Como cuando varios cocineros preparan un plato, pero no se puede servir hasta que todos terminen su parte.

**5. ¿Cómo verificarías que realmente se está ejecutando en GPU y no en CPU?**

Revisaría nvidia-smi y también vería si el kernel aparece corriendo. Si la GPU sube de uso, entonces sí está funcionando.

**Josue Say**

**1. ¿Qué aprendiste sobre cómo CUDA distribuye el trabajo entre hilos y bloques?**

Aprendí que CUDA asigna bloques a diferentes multiprocesadores y dentro de cada uno ejecuta muchos hilos en paralelo. La estructura está pensada para dividir problemas grandes en tareas pequeñas.

**2. ¿Qué fue lo más difícil de entender del paralelismo?**

Lo que más me costó fue comprender cómo se coordinan los hilos y por qué no se pueden sincronizar bloques completos. También la idea de que el orden de salida nunca es determinista.

**3. Si pudieras mejorar el laboratorio, ¿qué cambio harías en el algoritmo?**

Implementaría una reducción más eficiente y también permitiría comparar el comportamiento entre diferentes configuraciones de hilos y bloques.

**4. ¿Qué analogía del mundo real usarías para explicar el concepto de “sincronización de hilos”?**

La compararía con un semáforo peatonal: todos esperan la luz verde para moverse exactamente al mismo tiempo.

**5. ¿Cómo verificarías que realmente se está ejecutando en GPU y no en CPU?**

Usaría cudaGetDeviceProperties, mediría tiempos de ejecución y verificaría uso de la GPU en nvidia-smi.

**Carlos Valladares**

**1. ¿Qué aprendiste sobre cómo CUDA distribuye el trabajo entre hilos y bloques?**

Aprendí a que cada bloque es como una unidad grande de trabajo y cada hilo hace una parte pequeña del cálculo, repartiendo esos bloques entre la GPU y cada hilo trabaja en paralelo con su propio índice.

**2. ¿Qué fue lo más difícil de entender del paralelismo?**

Lo más difícil de entender fue que el orden no está garantizado y que todos los hilos trabajan simultáneamente, y entender que solo se pueden sincronizar hilos dentro del mismo bloque.

**3. Si pudieras mejorar el laboratorio, ¿qué cambio harías en el algoritmo?**

Agregaría una parte donde las galaxias puedan compararse entre ellas usando una reducción global o una visualización para ver los brillos de forma más clara.

**4. ¿Qué analogía del mundo real usarías para explicar el concepto de “sincronización de hilos”?**

El mejor ejemplo sería cuando en clase todos deben terminar un ejercicio antes de continuar con otro, no importa que unos terminen rápido y otros lento, nadie va avanzar hasta que todos estén listos.

**5. ¿Cómo verificarías que realmente se está ejecutando en GPU y no en CPU?**

Revisando nvidia-smi y usando cudaGetDeviceProperties comparando los tiempos con una versión de CPU y verificando que el kernel se lance sin errores desde la GPU.

**Respuestas conjuntas**

**1. ¿Qué aprendiste sobre cómo CUDA distribuye el trabajo entre hilos y bloques?**

Como grupo llegamos a la conclusión de que CUDA organiza el trabajo dividiéndolo en bloques y dentro de cada bloque en hilos. Esto permite repartir tareas grandes en muchas tareas pequeñas que se ejecutan al mismo tiempo. También entendimos que cada hilo usa su propio índice para calcular algo diferente y que los bloques se distribuyen entre los multiprocesadores de la GPU.

**2. ¿Qué fue lo más difícil de entender del paralelismo?**

Coincidimos en que lo más complicado fue acostumbrarnos a que no existe un orden fijo y que los hilos ejecutan su trabajo simultáneamente. Además, todos tuvimos dificultades al inicio para entender por qué únicamente se pueden sincronizar hilos dentro de un mismo bloque y no entre bloques completos.

**3. Si pudieras mejorar el laboratorio, ¿qué cambio harías en el algoritmo?**



Como grupo pensamos que sería útil agregar una parte donde las galaxias puedan compararse entre sí o mostrar alguna visualización del comportamiento de los brillos. También consideramos que sería interesante implementar una reducción global o probar diferentes configuraciones de hilos y bloques para ver cómo cambia el rendimiento.

**4. ¿Qué analogía del mundo real usarías para explicar el concepto de “sincronización de hilos”?**

Llegamos a la analogía de que la sincronización es como cuando un grupo de personas tiene que esperar a que todos estén listos para seguir con la siguiente actividad. No importa si unos terminan antes y otros después, nadie puede avanzar hasta que todos estén preparados.

**5. ¿Cómo verificarías que realmente se está ejecutando en GPU y no en CPU?**

Como grupo concluimos que los métodos más confiables son revisar el uso de la GPU con `nvidia-smi`, comprobar las propiedades del dispositivo con `cudaGetDeviceProperties`, y comparar la velocidad del programa frente a una versión que corra en CPU. Si el kernel aparece ejecutándose y la GPU muestra actividad, significa que sí está corriendo en la GPU.