Advanced CUDA Topics: Dynamic Parallelism

Slides adapted from Nvidia Devblog

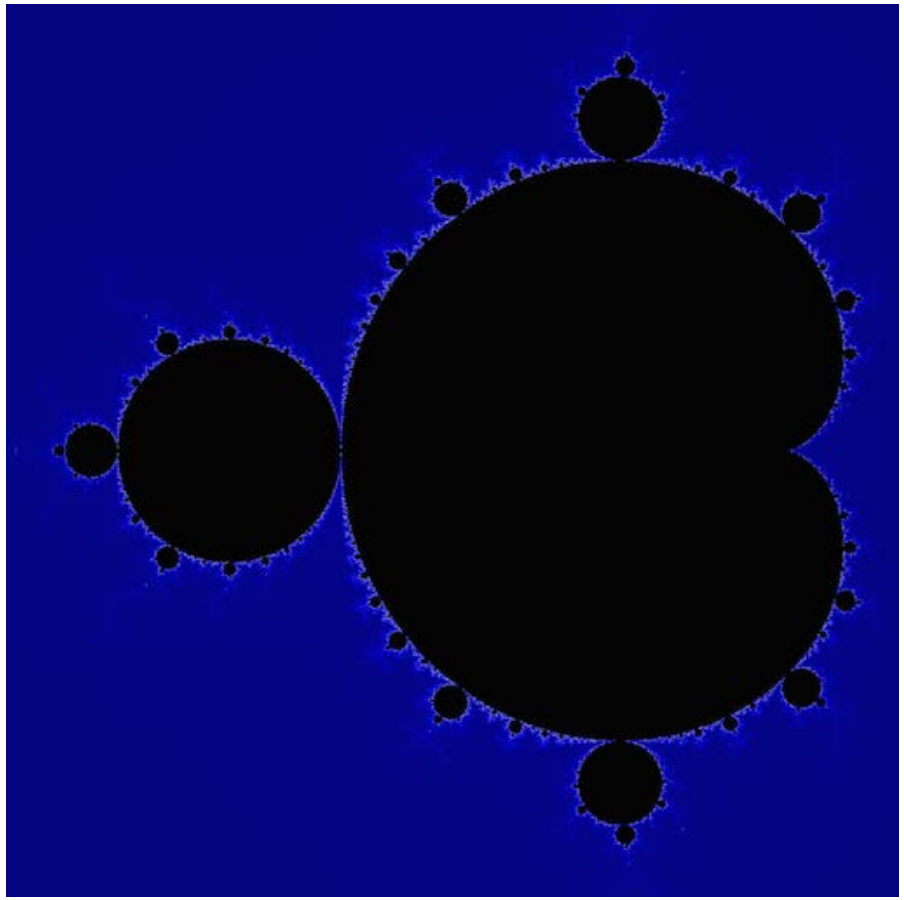
Philipp Gschwandtner

Recap: Host-to-device Interaction is Slow

- “To get good performance we have to live on the GPU.” (Lukas Einkemmer, March 10th 2020)
- But what about highly dynamic problems that require creating new work on-demand?

Case Study: Mandelbrot Set

- well-known fractal with a very simple mathematical definition
- $z_0 = c$
- $z_{n+1} = z_n^2 + c$
- $M = \{c \in \mathbb{C} : \exists\, R\ \forall\, n : |z_n| < R\}$

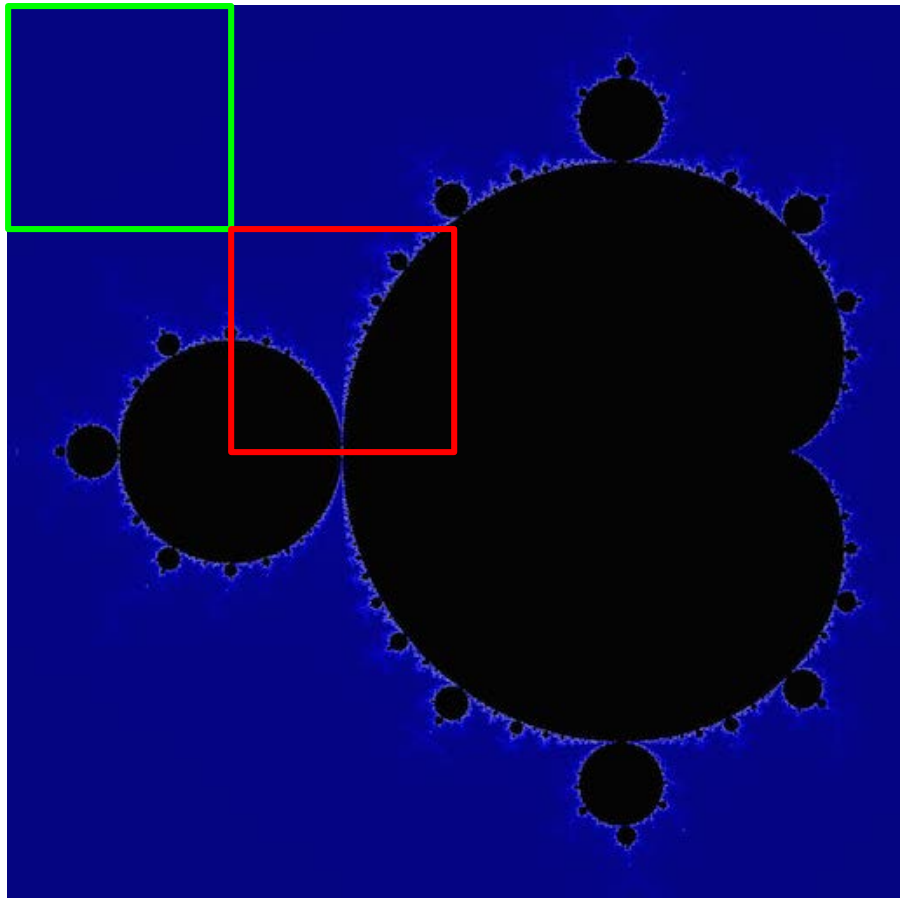


Simple Algorithm: “Time Escape”

```
for each pixel p
  dwell = 0
  c = toComplex(p.x, p.y)
  z = c
  while dwell < MAX_DWELL and abs(z) < 2*2
    z = z * z + c;
    dwell++;
  end
  color[p] = getColor(dwell);
end
```

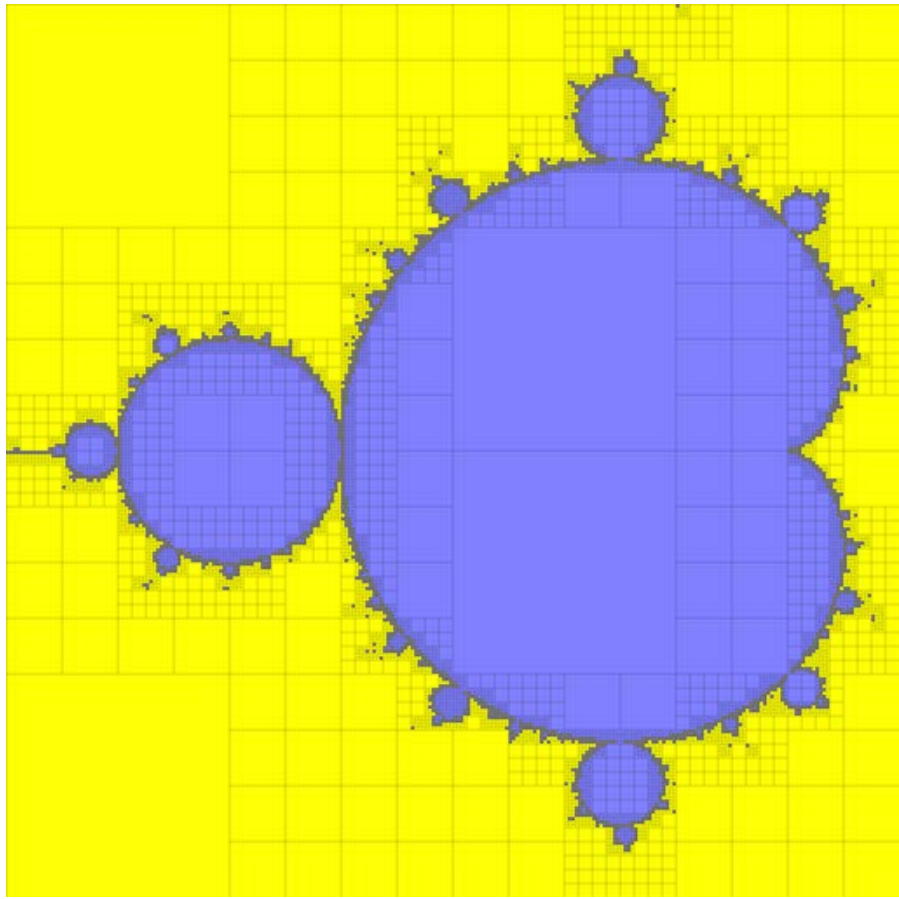
Simple Algorithm: “Time Escape” cont’d

- This is comparatively fast…
  - 34 ms for a 4096x4096 image with a MAX_DWELL of 1024 @ Tesla V100
- But it is also quite inefficient
  - Large areas of pixels are apparently all inside or outside the set
  - Inefficient expense of resources
  - Knowing the shape of Mandelbrot, can we improve the efficiency?



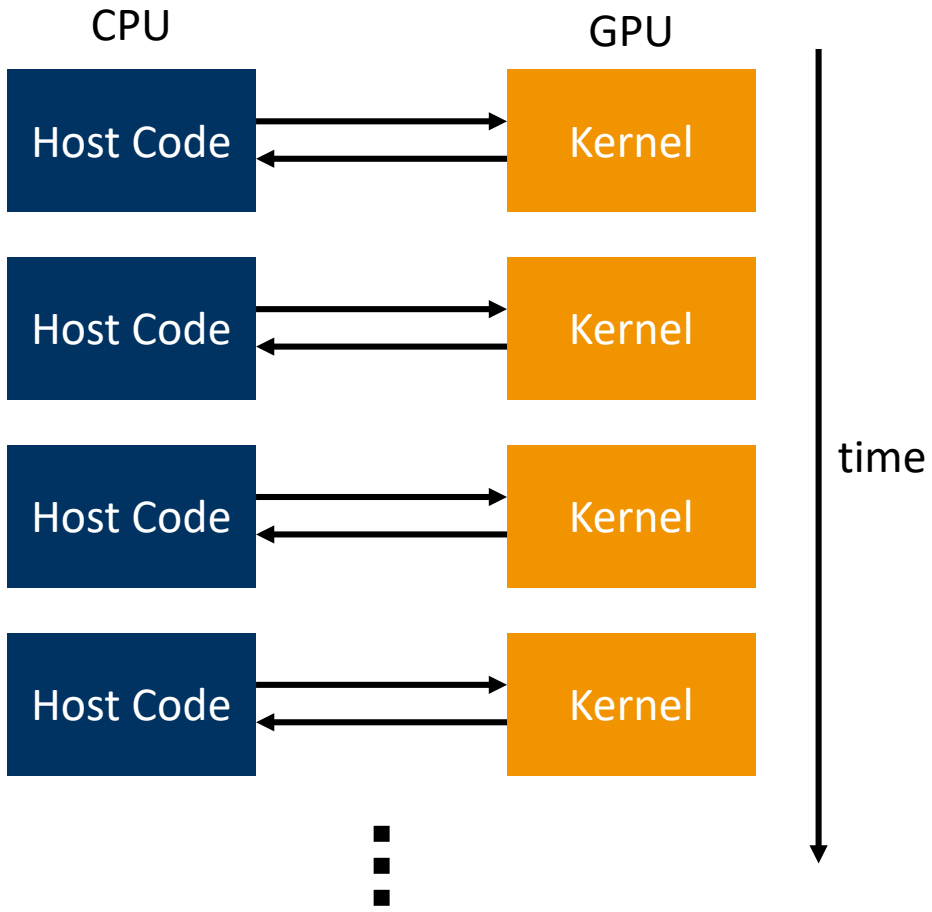
Adaptive Grid Refinement

- Mandelbrot set is *connected*
  - there is a path between any two points in the set
  - if the border of any region is entirely inside or outside the set, the entire region is inside or outside the set
- Idea: form large cells, check border first, only refine and spend more work if required



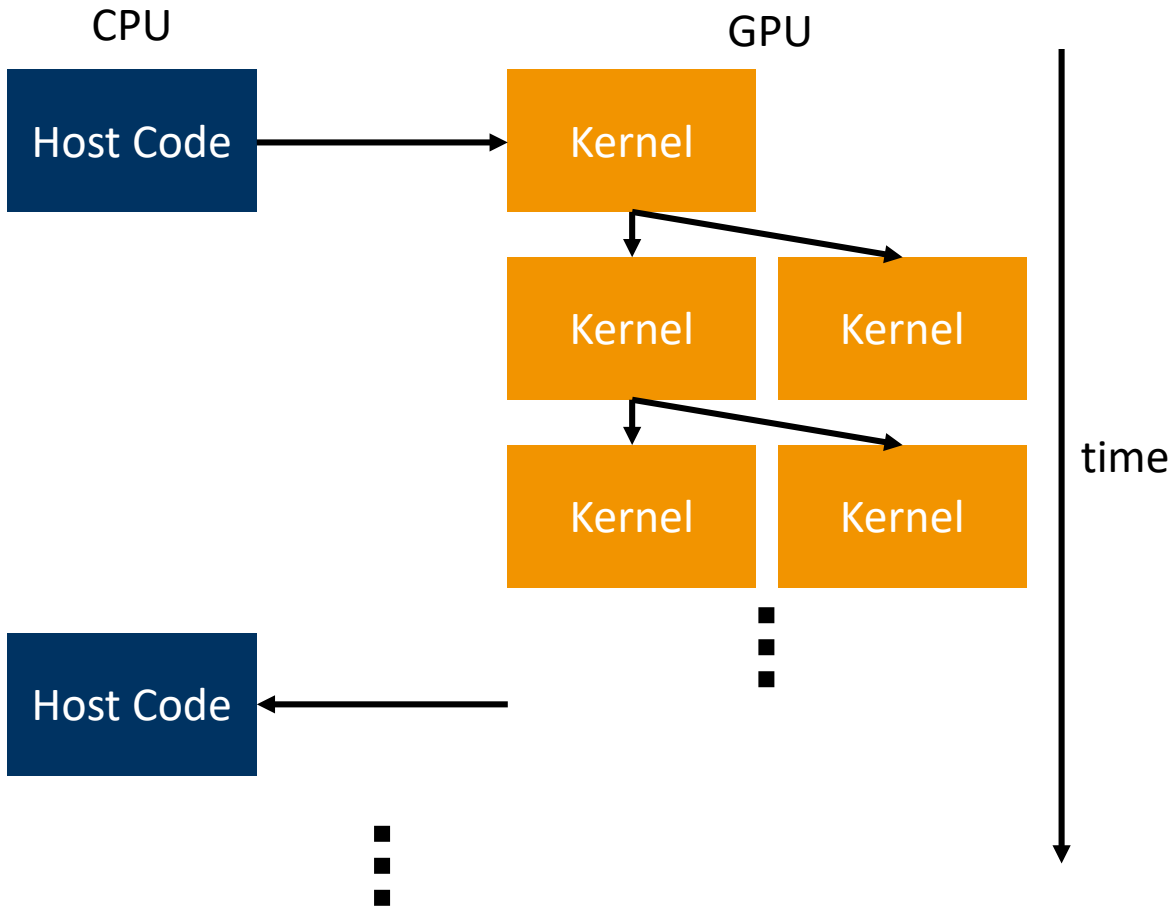
Problem

- Inefficient execution model
  - CPU launches kernel
  - GPU evaluates if new kernels should be launched and reports back to CPU
  - CPU launches new kernel
  - Repeat until done



Solution: Dynamic Parallelism

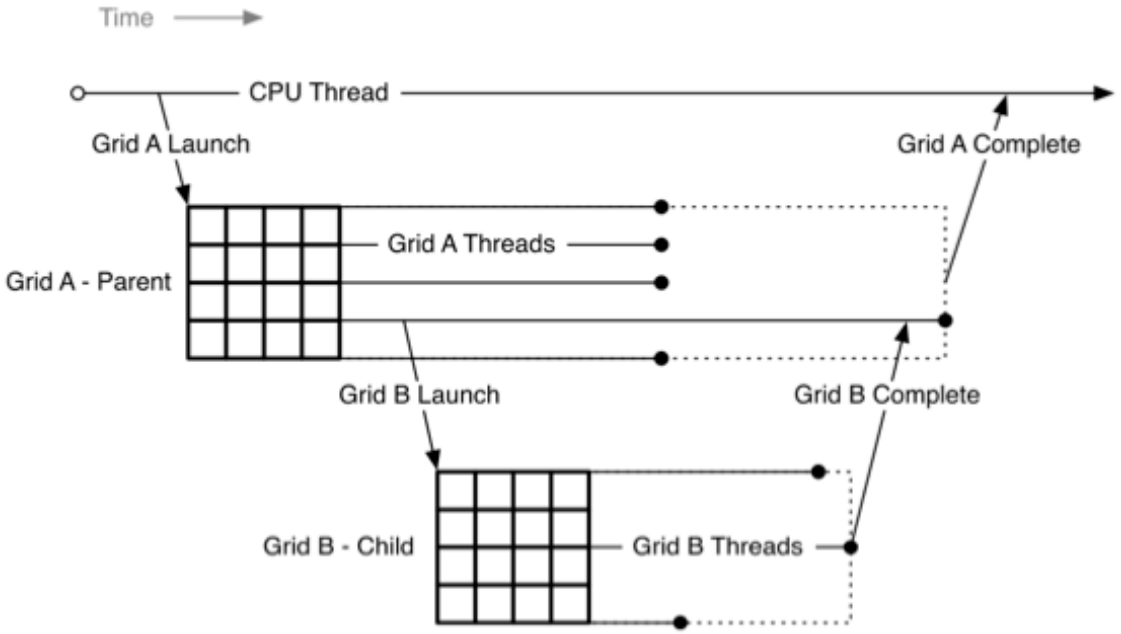
- GPU can launch new kernels
  - Reduces dependency on CPU
  - Improves kernel throughput
- Enables dynamic workloads without performance penalty
- Available with compute capability ≥ 3.5



Grid Nesting and Synchronization

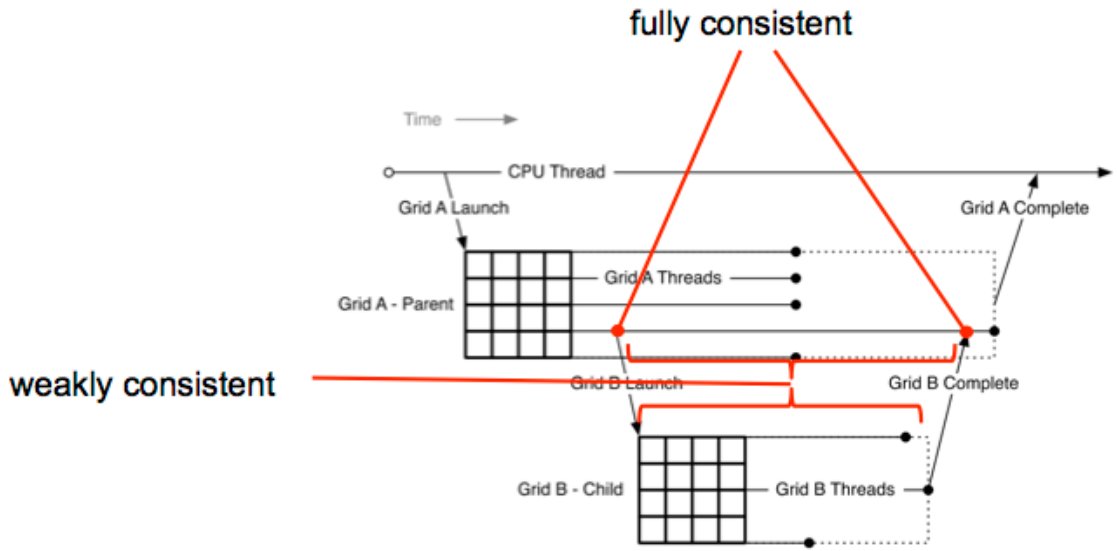
- Grid launches are fully nested
  - Child grids always complete before parent that launched them (no explicit synchronization required)
  - If child grid results are required in parent use `cudaDeviceSynchronize()`

- Note: *grid* is CUDA speak for a group of blocks of threads running a kernel



Memory Consistency

- Global memory consistency guaranteed upon child start and end
  - Prior parent grid writes are visible to child upon launch
  - Child writes are visible to parent after child completed
- But no global memory guarantees in-between!



Example Code

- Code on the right shows nested kernel invocation
  - But has a race condition!
  - `child_k()` might print “1” or “2”

```
__device__ int v = 0;

__global__ void child_k(void) {
  printf("v = %d\n", v);
}

__global__ void parent_k(void) {
  v = 1;
  child_k <<< 1, 1 >>>> ();
  v = 2;
  cudaDeviceSynchronize();
}
```

Compile Flags

- Compile with compute capability 3.5 or higher, e.g.
  - `-arch=sm_35`
- Generate relocatable device code for linking
  - `-rdc=true`
- Link against cuda device runtime
  - `-lcudadevrt`

Caveats

- `cudaDeviceSynchronize()` is expensive, use only when required

- Careful when passing pointers to child grids
  - ✅ global memory, zero-copy host memory, constant memory
  - ❌ shared memory, local memory (incl. stack variables)

- Careful with number of recursive kernel launches
  - Limits for nesting kernel calls
  - Limits for nesting `cudaDeviceSynchronize()`
  - May affect correctness and performance

Practical Exercise: Mandelbrot

- Goal: Using dynamic parallelism in a “real” application and implement the decision-making in an adaptive refinement algorithm
- Examine `day_4/ho3/mandelbrot-dyn.cu`
  - `mandelbrot.cu` shows a non-adaptive implementation
  - Requires open-source library libpng (installation provided on Innsbruck cluster)
- Complete the missing kernel calls
  - Positions marked with `// ASSIGNMENT`
- Output of both adaptive and non-adaptive implementations should be the same
  - But compare performance!

Conclusion

- Dynamic parallelism can increase performance of computationally highly dynamic problems
- Performance improvements up to several factors depending on use case
- Always be aware of amount of synchronization required and number of nested kernel launches

Sources

- https://devblogs.nvidia.com/cuda-dynamic-parallelism-api-principles/
- https://devblogs.nvidia.com/introduction-cuda-dynamic-parallelism/