Tales from NVIDIA

Seattle, Washington, USA

Seattle
Washington
Montana
North Dakota
Minnesota
Quebec
New Brunsw
Portland
Ottawa
Montreal
Minneapolis
Maine
South Dakota
Wisconsin
Vermont
Oregon
Idaho
Michigan
Toronto
Wyoming
Milwaukee
New Hampshire
Hamilton
New York
Detroit
Massachusetts
Chicago
Iowa
Cleveland
Connecticut
Nebraska
Ohio
Pennsylvania
New York
Illinois
Indiana
Philadelphia
Denver
Nevada
United States
Columbus
Maryland
New Jersey
Kansas City
Indianapolis
Utah
St. Louis
West Virginia
Washington
Sacramento
Colorado
Cincinnati
Delaware
Kansas
Missouri
San Francisco
San Jose
California
Kentucky
Virginia
Las Vegas
Tennessee
North Carolina
Oklahoma
Albuquerque
Arkansas
Charlotte
Los Angeles
Arizona
Atlanta
South Carolina
New Mexico
Mississippi
San Diego
Phoenix
Dallas
Alabama
Georgia
Tucson
Ciudad Juarez
Baja California
Texas
Jacksonville
Sonora
Louisiana
Houston
San Antonio
Chihuahua
Coahuila de Zaragoza
Orlando
Tampa
Florida
Satellite
Gulf of California
Nuevo León
Google
Miami

WINTER: 46 F, grey, gloomy, (8° C) probably raining.

(Basically a Tim Burton movie)

http://theoatmeal.com/blog/seattle_weather

SPRING: (See winter)

http://theoatmeal.com/blog/seattle_weather

SUMMER:

Finally shows up in late July.

The whole city gets all manic about how AMAAAAAAZING our weather is

http://theoatmeal.com/blog/seattle_weather

2 SECONDS LATER

blog/seattle_weather


BNSF

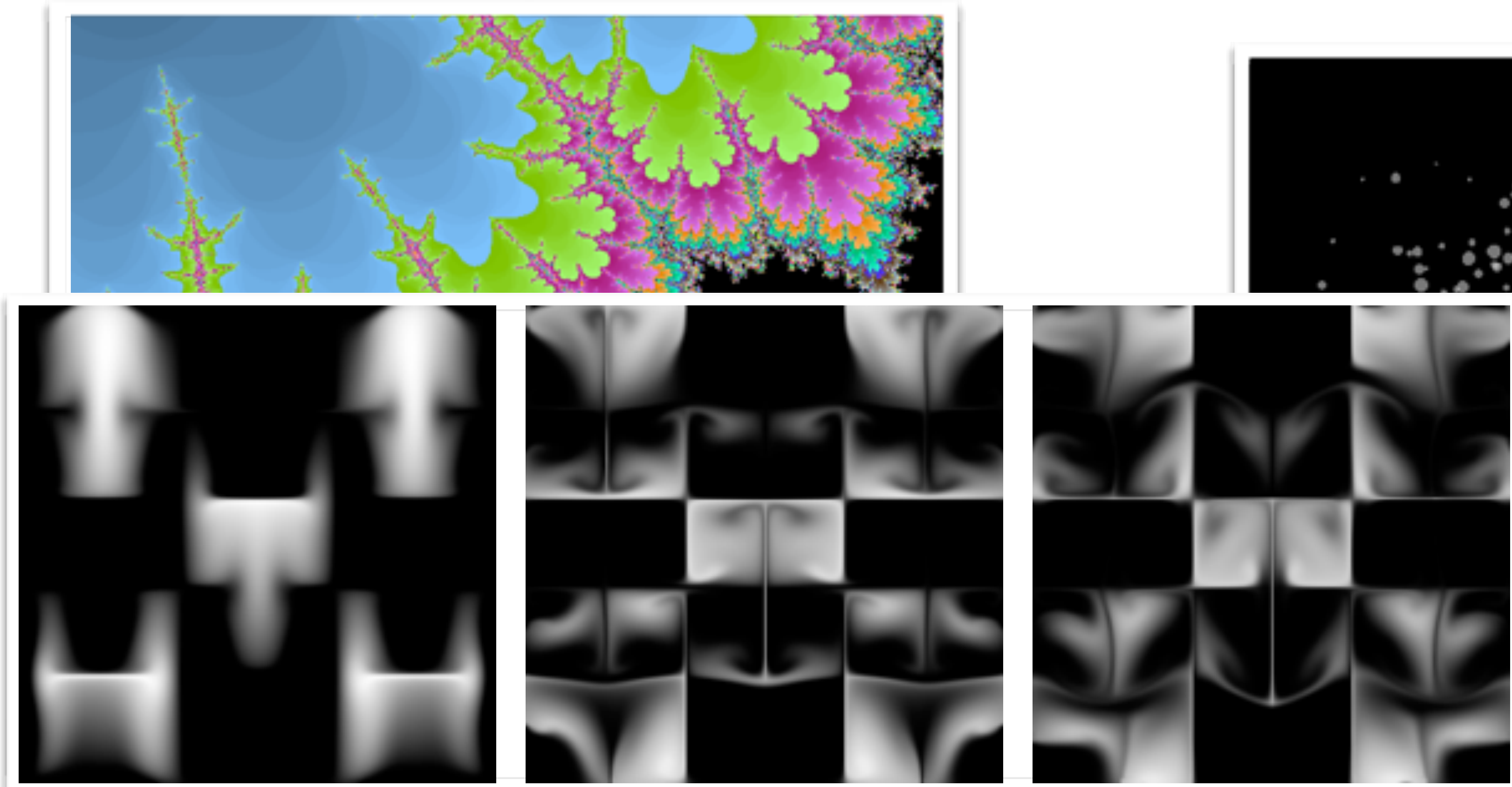

BNSF

An embedded language for CPU and GPU metaprogramming

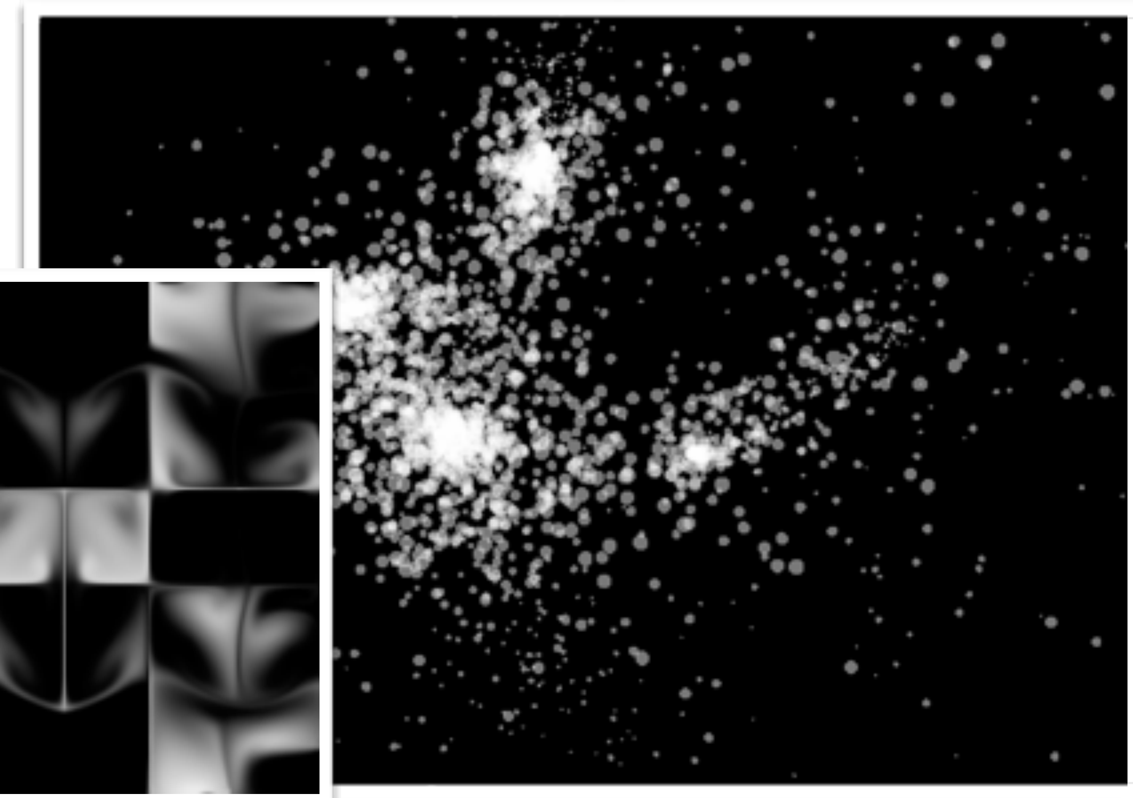
Trevor L. McDonell
University of New South Wales, Australia

Jointly with
Vinod Grover
Sean Lee

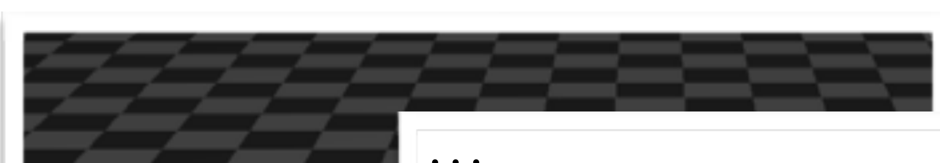
tlmcdonell
tmcdonell



stable fluid flow



n-body gravitational simulation

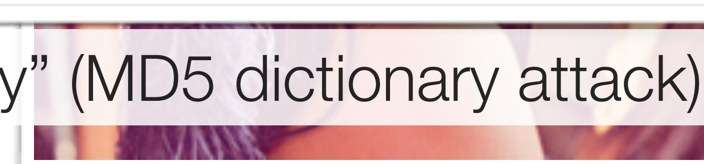


```
...
d6b821d937a4170b3c4f8ad93495575d: saitek1
d0e52829bf7962ee0aa90550ffdcccaa: laura1230
494a8204b800c41b2da763f9bbbcc462: lina03
d8ff07c52a95b30800809758f84ce28c: Jenny10
e81bed02faa9892f8360c705241191ae: carmen89
46f7d75718029de99dd81fd907034bc9: mellon22
0dd3c176cf34486ec00b526b6920b782: helena04
9351c4bc8c8ba17b58d5a6a1f839f356: 85548554
9c36c5599f40d08f874559ac824d091a: 585123456
4b4dce6c91b429e8360aa65f97342e90: 5678go
3aa561d4c17d9d58443fc15d10cc86ae: momo55

Recovered 150/1000 (15.00 %) digests in 59.45 s, 185.03 MHash/sec
```

SmoothLife cellular automata

Password “recovery” (MD5 dictionary attack)



Canny edge detection

Accelerate

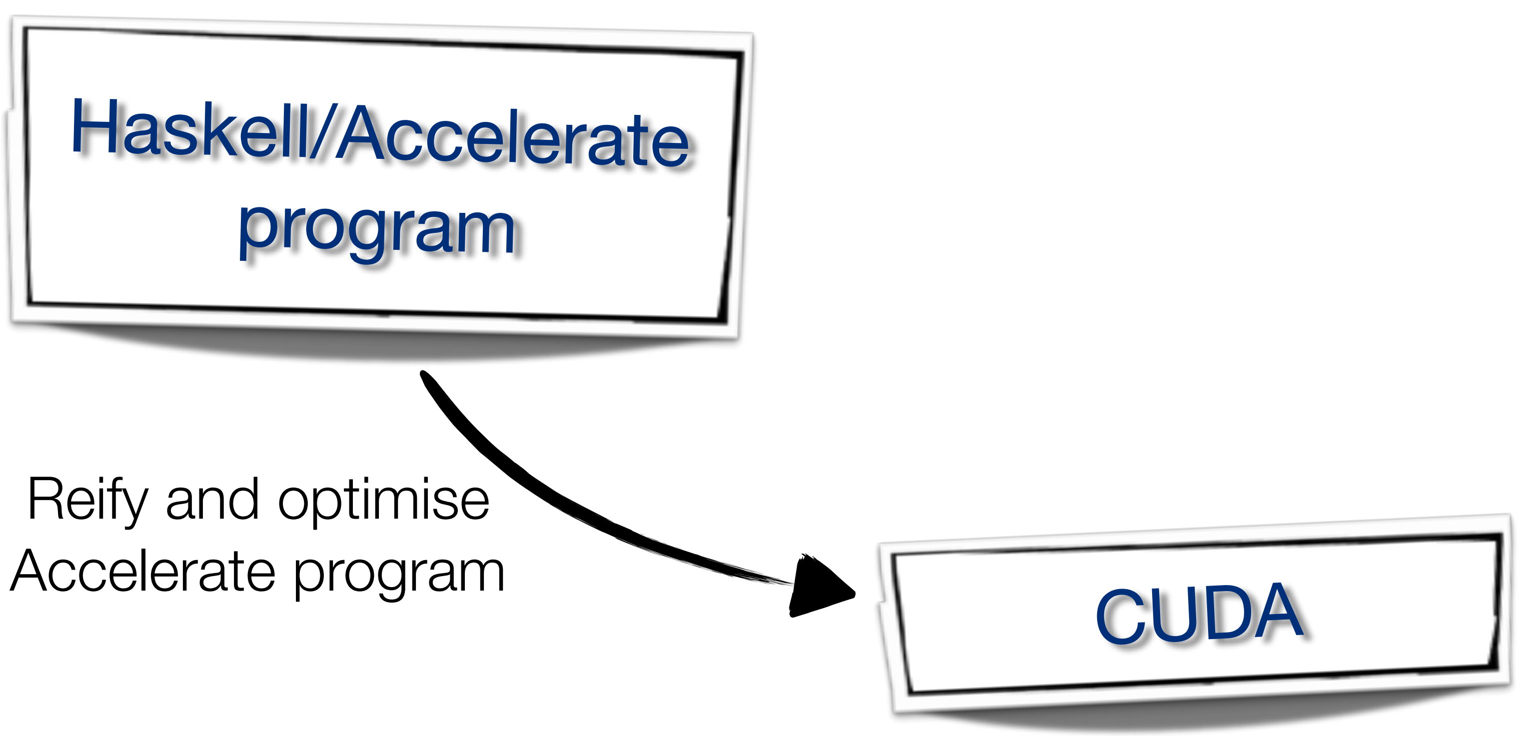
- Accelerate is a Embedded Domain-Specific Language for high-performance computing

Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing

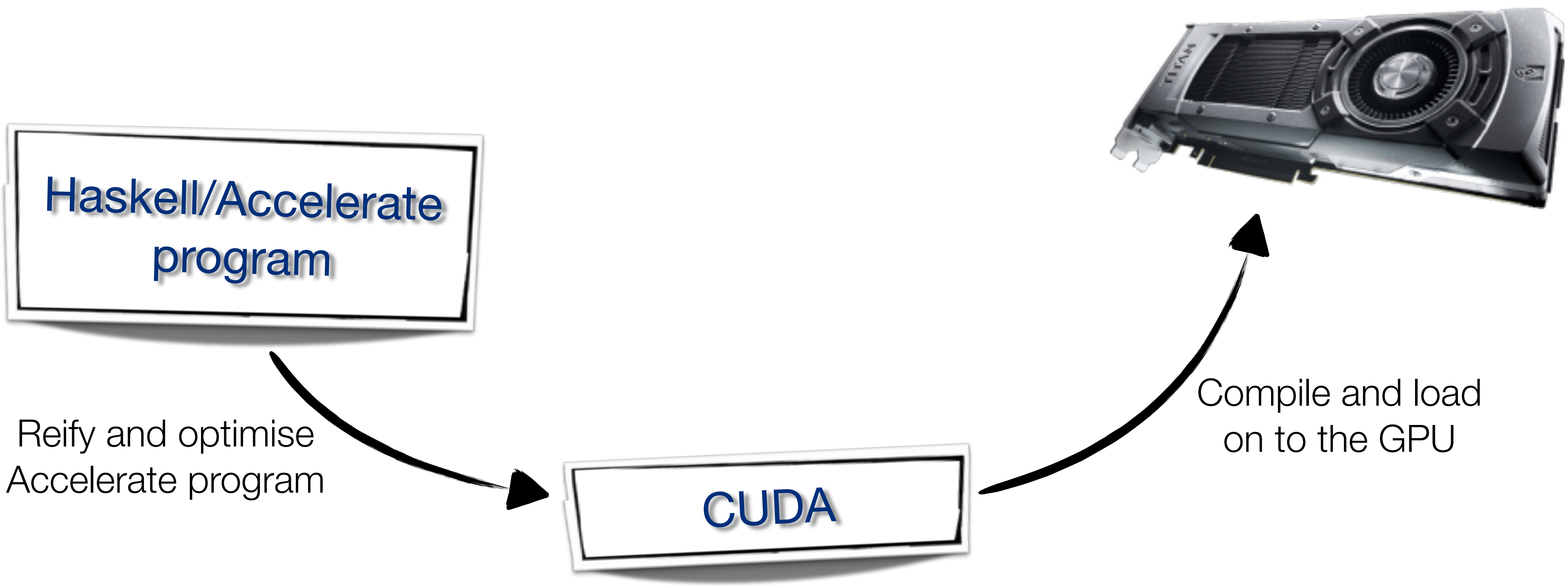
Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing



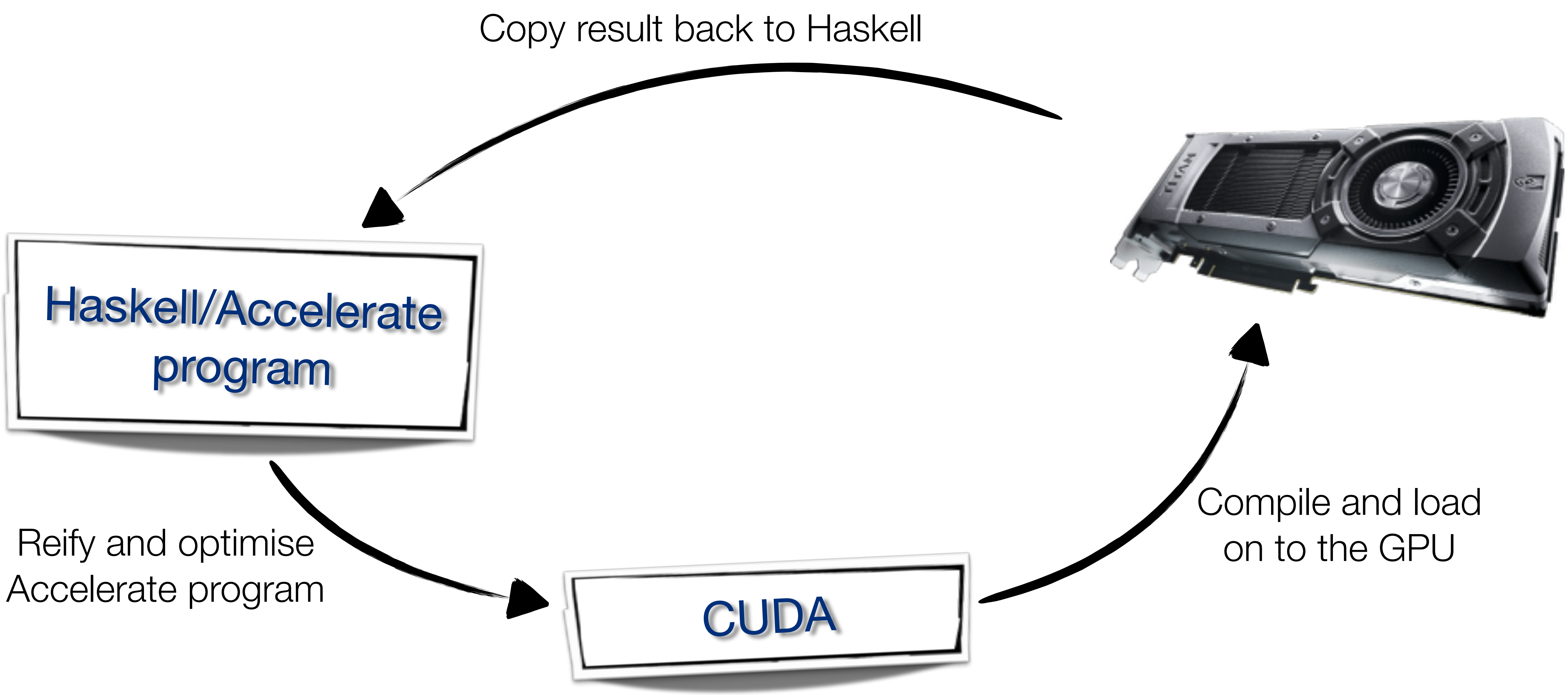
Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing



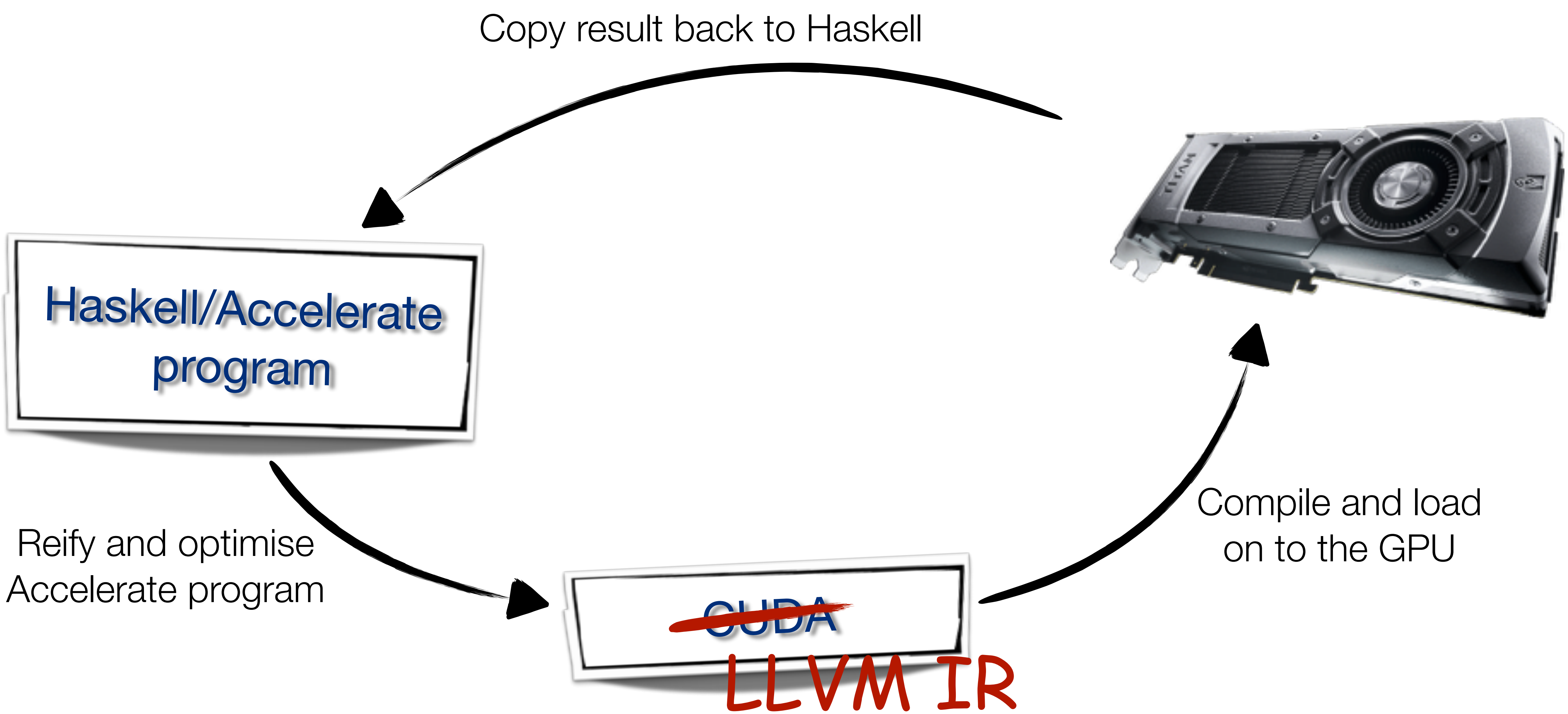
Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing



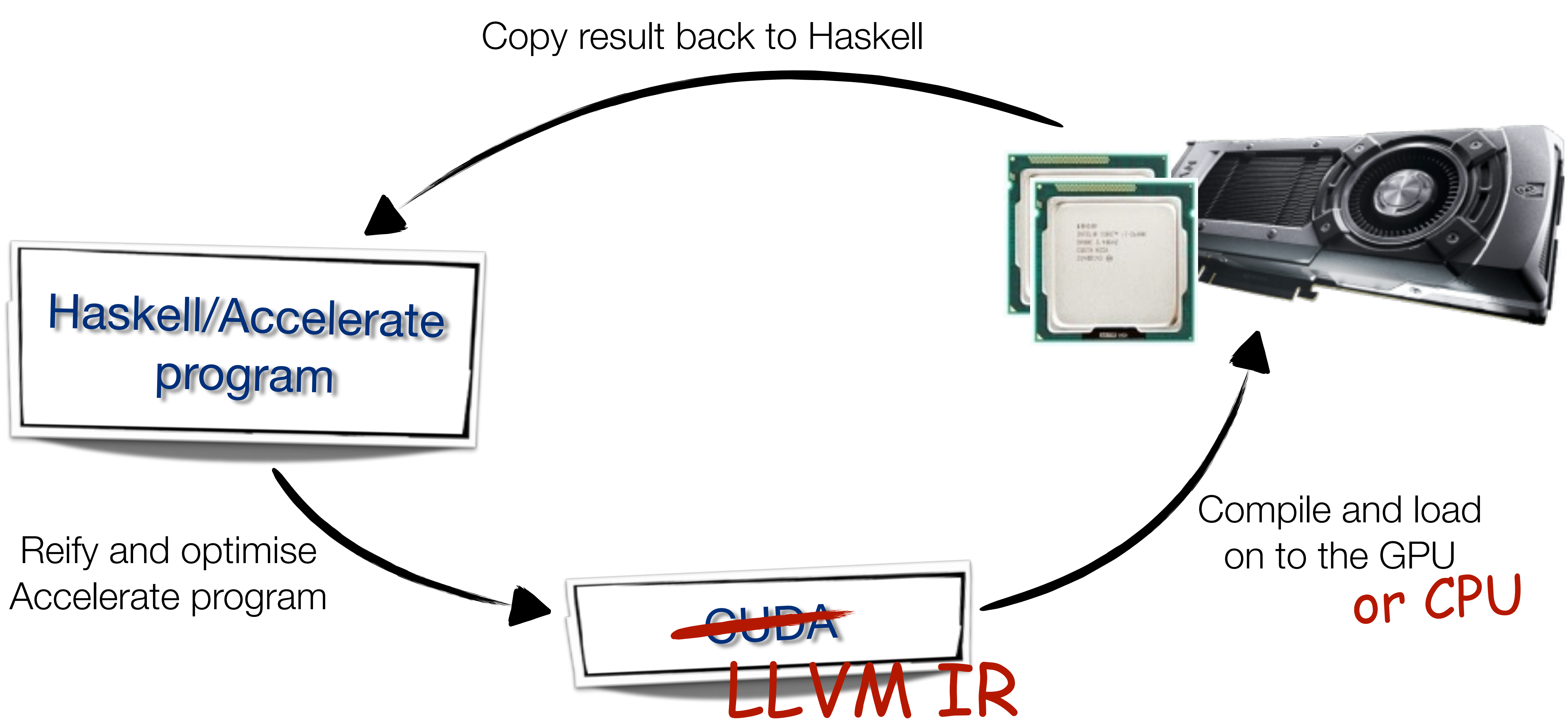
Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing



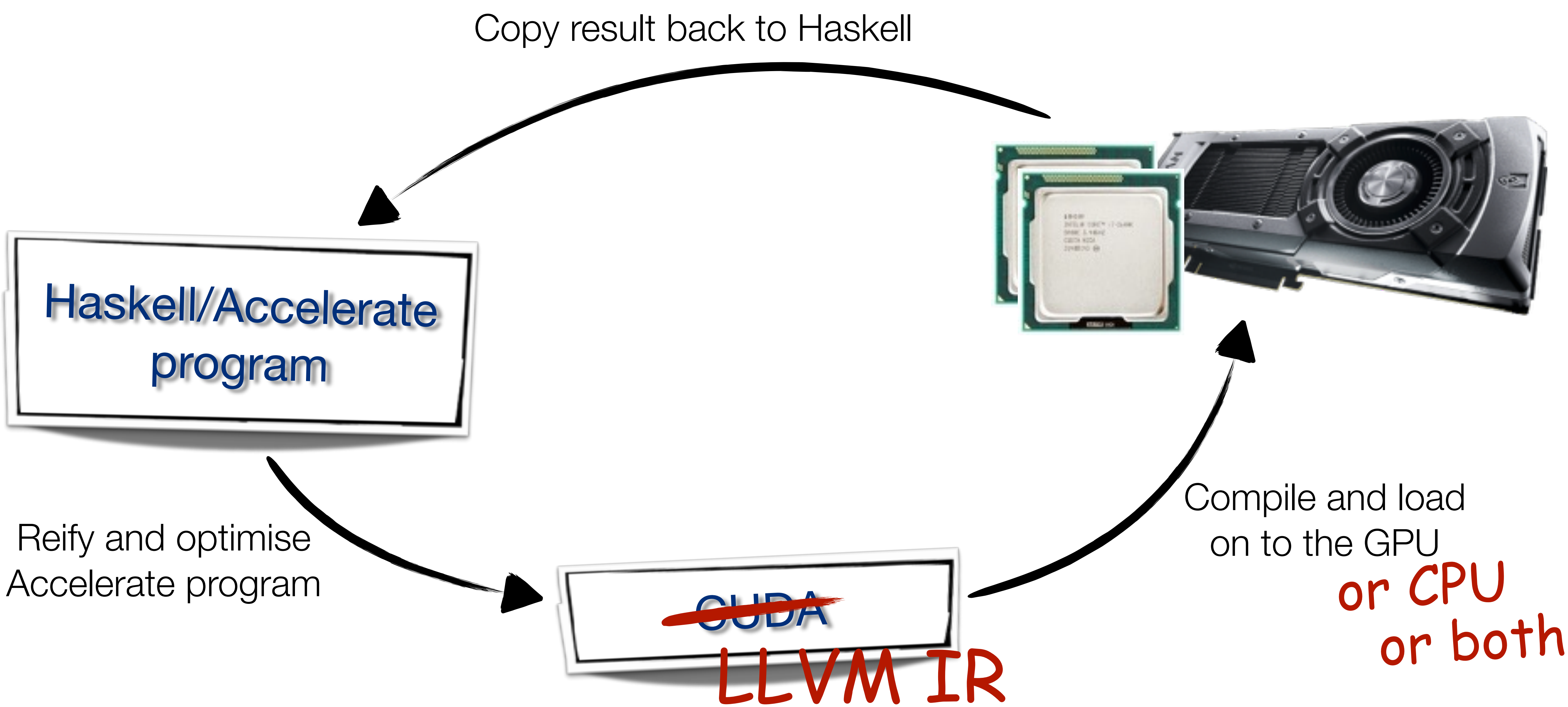
Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing



Accelerate

- Accelerate is a Embedded Domain-Specific Language for high-performance computing



Accelerate-LLVM backend

LLVM

- Compiler infrastructure project written for use by other compiler writers
  - Not intended for end users: low level representation
  - Includes optimisation and code generation support for many architectures, including x86* and NVIDIA GPUs
  - Supports online compilation

LLVM… in Accelerate

LLVM… in Accelerate

- Existing backend generates CUDA C code
    - But, calling *nvcc* from an online compiler is expensive

LLVM… in Accelerate

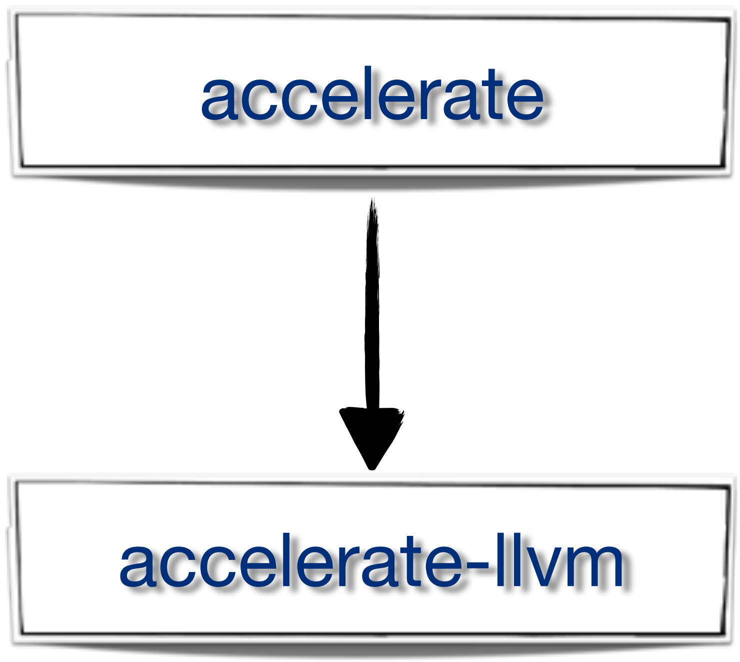
- Existing backend generates CUDA C code
  - But, calling *nvcc* from an online compiler is expensive
- IDEA: A new backend that generates LLVM IR
  - NVIDIA GPU code using NVPTX/libNVVM, execute with CUDA bindings
  - Vectorized x86 code, execute using machine-code JIT
  - Other targets possible: reuse and share functionality

Accelerate-LLVM

- Accelerate compiler infrastructure project

Accelerate-LLVM

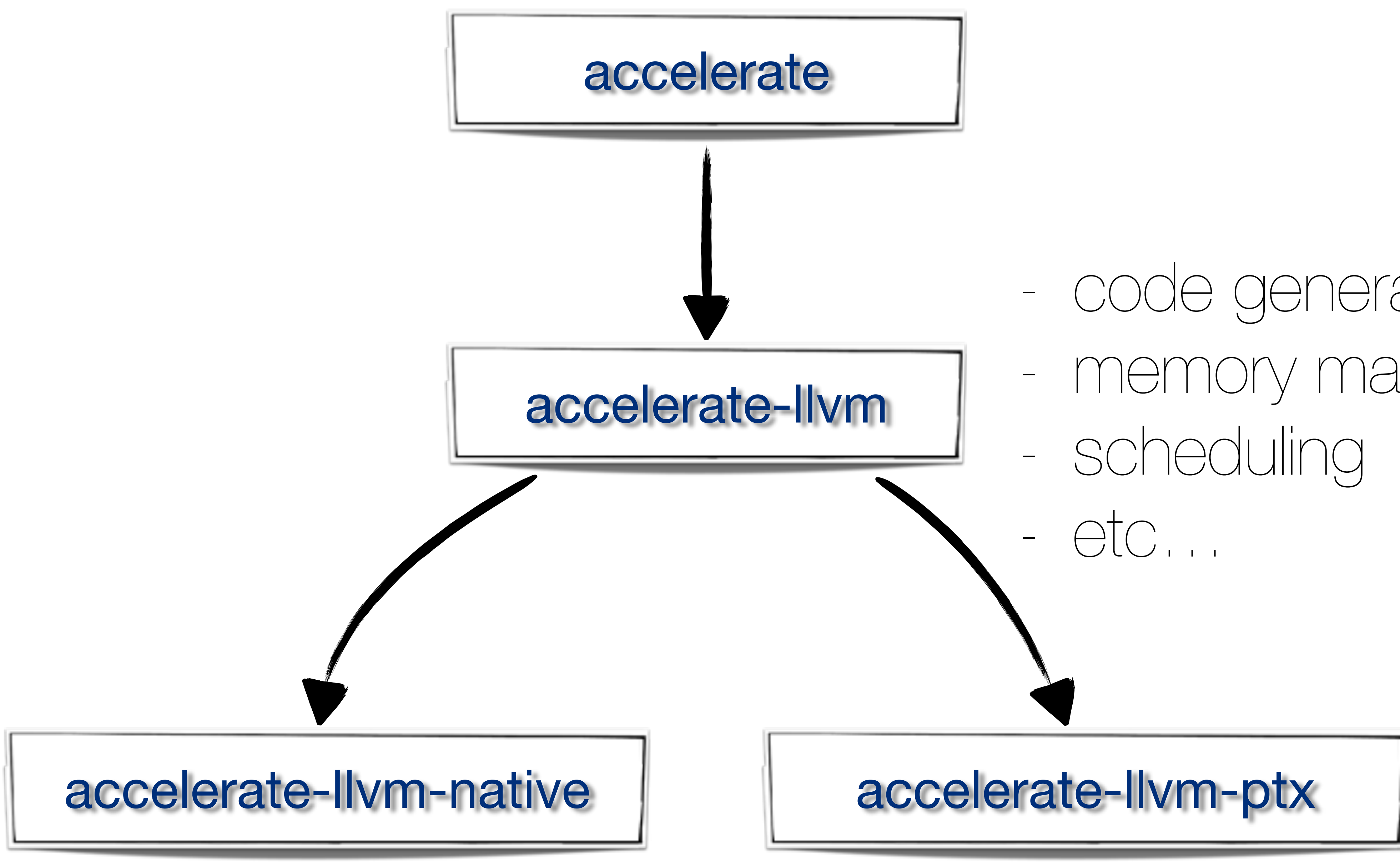
- Accelerate compiler infrastructure project



- code generation
- memory management
- scheduling
- etc…

Accelerate-LLVM

- Accelerate compiler infrastructure project



- code generation
- memory management
- scheduling
- etc…

- skeletons
- compilation
- etc…

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - operations are parameterised by the type of the backend Target
  - can contain target-specific state (caches, execution resources)

```
class Target arch where
  targetTriple          :: arch {- dummy -} → Maybe String
  targetDataLayout      :: arch {- dummy -} → Maybe DataLayout
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - operations are parameterised by the type of the backend Target
  - can contain target-specific state (caches, execution resources)

```
class Target arch where
  targetTriple           :: arch {- dummy -} → Maybe String
  targetDataLayout       :: arch {- dummy -} → Maybe DataLayout
```

```
data PTX = PTX {
      ptxContext           :: Context
    , ptxMemoryTable       :: MemoryTable
    , ptxStreamReservoir   :: Reservoir
  }

data Native = Native {
      nativeThreadGang     :: Gang
  }
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Code generation for scalar operations is (mostly) uniform, shared by all
  - Backends must specify how to instantiate each skeleton

```
class Skeleton arch where
  map   :: (Shape sh, Elt a, Elt b)
        => arch
        -> Gamma aenv
        -> IRFun1    aenv (a -> b)
        -> IRDelayed aenv (Array sh a)
        -> CodeGen [Kernel arch aenv (Array sh b)]
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Code generation for scalar operations is (mostly) uniform, shared by all
  - Backends must specify how to instantiate each skeleton

```
class Skeleton arch where
  map    :: (Shape sh, Elt a, Elt b)
         => arch
         -> Gamma aenv
         -> IRFun1    aenv (a -> b)
         -> IRDelayed aenv (Array sh a)
         -> CodeGen [Kernel arch aenv (Array sh b)]
```

generated LLVM IR

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Code generation for scalar operations is (mostly) uniform, shared by all
  - Backends must specify how to instantiate each skeleton

```
class Skeleton arch where
  map   :: (Shape sh, Elt a, Elt b)
        ⇒ arch
        → Gamma aenv
        → IRFun1    aenv (a → b)
        → IRDelayed aenv (Array sh a)
        → CodeGen [Kernel arch aenv (Array sh b)]
```

free array variables

generated LLVM IR

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Code generation for scalar operations is (mostly) uniform, shared by all
  - Backends must specify how to instantiate each skeleton

```
class Skeleton arch where
  map    :: (Shape sh, Elt a, Elt b)
         => arch
         -> Gamma aenv
         -> IRFun1    aenv (a -> b)
         -> IRDelayed aenv (Array sh a)
         -> CodeGen [Kernel arch aenv (Array sh b)]
```

generated LLVM IR

free array variables

LLVM global function

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Compile to the backend-specific [executable] format
  - Automatically caching code with the knot-tying trick (as used by run1)

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Compile to the backend-specific [executable] format
  - Automatically caching code with the knot-tying trick (as used by run1)

```
class Target arch ⇒ Compile arch where
  data ExecutableR t
  compileForTarget :: DelayedOpenAcc aenv a
                   → Gamma aenv
                   → LLVM arch (ExecutableR arch)
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
    - Compile to the backend-specific [executable] format
    - Automatically caching code with the knot-tying trick (as used by run1)

class-associated datatype

```
class Target arch ⇒ Compile arch where
  data ExecutableR t
  compileForTarget :: DelayedOpenAcc aenv a
                   → Gamma aenv
                   → LLVM arch (ExecutableR arch)
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Compile to the backend-specific [executable] format
  - Automatically caching code with the knot-tying trick (as used by run1)

class-associated datatype

```
class Target arch ⇒ Compile arch where
  data ExecutableR t
  compileForTarget :: DelayedOpenAcc aenv a
                   → Gamma aenv
                   → LLVM arch (ExecutableR arch)
```

```
instance Target PTX where
  data ExecutableR PTX = PTXR { ptxKernel :: [CUDA.Kernel]
                              , ptxModule :: CUDA.Module }

instance Target Native where
  data ExecutableR Native = NativeR { Function }
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Abstracts over AST traversals and the target type

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Abstracts over AST traversals and the target type

```
class Remote arch ⇒ Execute arch where
  map :: (Shape sh, Elt b)
      ⇒ ExecutableR arch
      → Gamma aenv
      → AvalR arch aenv
      → StreamR arch
      → sh
      → LLVM arch (Array sh b)
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Abstracts over AST traversals and the target type

memory management

```
class Remote arch ⇒ Execute arch where
  map :: (Shape sh, Elt b)
      ⇒ ExecutableR arch
      → Gamma aenv
      → AvalR arch aenv
      → StreamR arch
      → sh
      → LLVM arch (Array sh b)
```

Accelerate-LLVM

- A framework for implementing LLVM-based Accelerate backends
  - Abstracts over AST traversals and the target type

memory management

```
class Remote arch ⇒ Execute arch where
  map :: (Shape sh, Elt b)
      ⇒ ExecutableR arch
      → Gamma aenv
      → AvalR arch aenv
      → StreamR arch
      → sh
      → LLVM arch (Array sh b)
```

asynchronous operations

Accelerate-LLVM

Accelerate-LLVM

- A collection of reusable components
  - Functionality provided by target-parameterised classes
  - Associated data-types for backend specific features

Accelerate-LLVM

- A collection of reusable components
  - Functionality provided by target-parameterised classes
  - Associated data-types for backend specific features

- Backends just specify what to do with each collective operation
  - CUDA backend: 9500 LOC
  - LLVM backend:
    - Base framework: 5400 LOC
    - Native backend*: 2400 LOC
    - PTX backend*: 4600 LOC

*not all operations supported

Implementation details

… & other dirty little secrets

Code generation

- Code generation uses the LLVM C/C++ API (via `llvm-general`)
- Generates clean, optimised LLVM directly in SSA
  - No stack allocation of mutable variables (alloca instruction)
  - Branches and loops use phi nodes
  - Adds appropriate annotations (`NoUnwind`, `NoAlias`, etc…)
  - Monadic interface to generating LLVM IR
- Skeletons are designed to allow LLVM auto-vectorisation (native target)
  - Generates SSE/AVX instructions for maps, folds, etc.

Code generation

- For GPU, supports compilation by both NVPTX and libNVVM
  - NVPTX: open source component of LLVM
  - libNVVM: closed source optimiser which is part of the CUDA toolkit
- Tension
  - libNVVM requires llvm == 3.2; but
  - Auto-vectorisation requires llvm >= 3.3

Executing x86

- The native backend lowers the LLVM IR into machine code
  - Crossing the FFI barrier into the LLVM API entails foreign state
  - LLVM-General API brackets creation and destruction of FFI calls: can not return anything from the continuation that depends on the foreign object

Executing x86

- The native backend lowers the LLVM IR into machine code
  - Crossing the FFI barrier into the LLVM API entails foreign state
  - LLVM-General API brackets creation and destruction of FFI calls: can not return anything from the continuation that depends on the foreign object

Required to compile to machine code

```
withHostTargetMachine :: (TargetMachine -> IO a) -> ErrorT String IO a
```

Executing x86

- The native backend lowers the LLVM IR into machine code
  - Crossing the FFI barrier into the LLVM API entails foreign state
  - LLVM-General API brackets creation and destruction of FFI calls: can not return anything from the continuation that depends on the foreign object

Required to compile to machine code

```
withHostTargetMachine :: (TargetMachine → IO a) → ErrorT String IO a
```

must not depend on the TargetMachine

Executing x86

- Capture compiled foreign functions into a worker thread

```
data Req = ReqDo (IO ()) | ReqShutdown

data Function = Function {
    functionTable      :: [(String, FunPtr ())]
  , functionReq        :: MVar Req
  , functionResult     :: MVar ()
  }
```

Executing x86

- Capture compiled foreign functions into a worker thread
  - Tell the thread to execute an action by writing it into the Req var

```haskell
data Req = ReqDo (IO ()) | ReqShutdown

data Function = Function {
    functionTable       :: [(String, FunPtr ())]
  , functionReq         :: MVar Req
  , functionResult      :: MVar ()
  }
```

thread is woken up when Req is filled

Executing x86

- Capture compiled foreign functions into a worker thread
  - Tell the thread to execute an action by writing it into the Req var
  - Wait for it to finish by reading from the result var

```
data Req = ReqDo (IO ()) | ReqShutdown

data Function = Function {
    functionTable      :: [(String, FunPtr ())]
  , functionReq        :: MVar Req
  , functionResult     :: MVar ()
  }
```

thread is woken up
when Req is filled

signal caller on completion

Executing x86

- Capture compiled foreign functions into a worker thread
  - Tell the thread to execute an action by writing it into the Req var
  - Wait for it to finish by reading from the result var
  - A finaliser on the `Function` sends `ReqShutdown` automatically on GC

```
data Req = ReqDo (IO ()) | ReqShutdown

data Function = Function {
    functionTable      :: [(String, FunPtr ())]
  , functionReq        :: MVar Req
  , functionResult     :: MVar ()
  }
```

thread is woken up when Req is filled

signal caller on completion

Executing x86

- The Function object executes the compiled LLVM executable
  - Communicating via `MVars` has some overhead

```
compileForNativeTarget acc aenv = do
  …
  fun ← startFunction $ \loop →
    withContext                      $ \ctx →
    …
    withModuleInEngine mcjit mdl $ \exe → do
        funs ← getGlobalFunctions ast exe
        loop funs
```

```
startFunction
    :: (([(String, FunPtr ())] → IO ()) → IO ())
    → IO Function
```

Executing x86

- The Function object executes the compiled LLVM executable
  - Communicating via `MVars` has some overhead

seven!

```
compileForNativeTarget acc aenv = do
  …
  fun ← startFunction $ \loop →
    withContext                       $ \ctx →
    …
    withModuleInEngine mcjit mdl $ \exe → do
        funs ← getGlobalFunctions ast exe
        loop funs
```

```
startFunction
    :: (([(String, FunPtr ())] → IO ()) → IO ())
    → IO Function
```

Executing x86

- The Function object executes the compiled LLVM executable
  - Communicating via `MVars` has some overhead

```
compileForNativeTarget acc aenv = do
  …
  fun ← startFunction $ \loop →
    withContext                  $ \ctx →
    …
    withModuleInEngine mcjit mdl $ \exe → do
        funs ← getGlobalFunctions ast exe
        loop funs
```

seven!

starts worker threads,
waits for requests

```
startFunction
    :: (([(String, FunPtr ())] → IO ()) → IO ())
    → IO Function
```

GPU memory management

- Require an association between host-side and device-side arrays
- Build a weak memory table from host side array to device side array
  - When the host array is GC'd, deallocate array and remove from the table

```
type MT c = MVar ( IntMap (RemoteArray c) )

data MemoryTable c = MemoryTable {
    memoryTable     :: MT c
  , memoryNursery   :: Nursery (c ())
  , weakTable       :: Weak (MT c)
  }
```

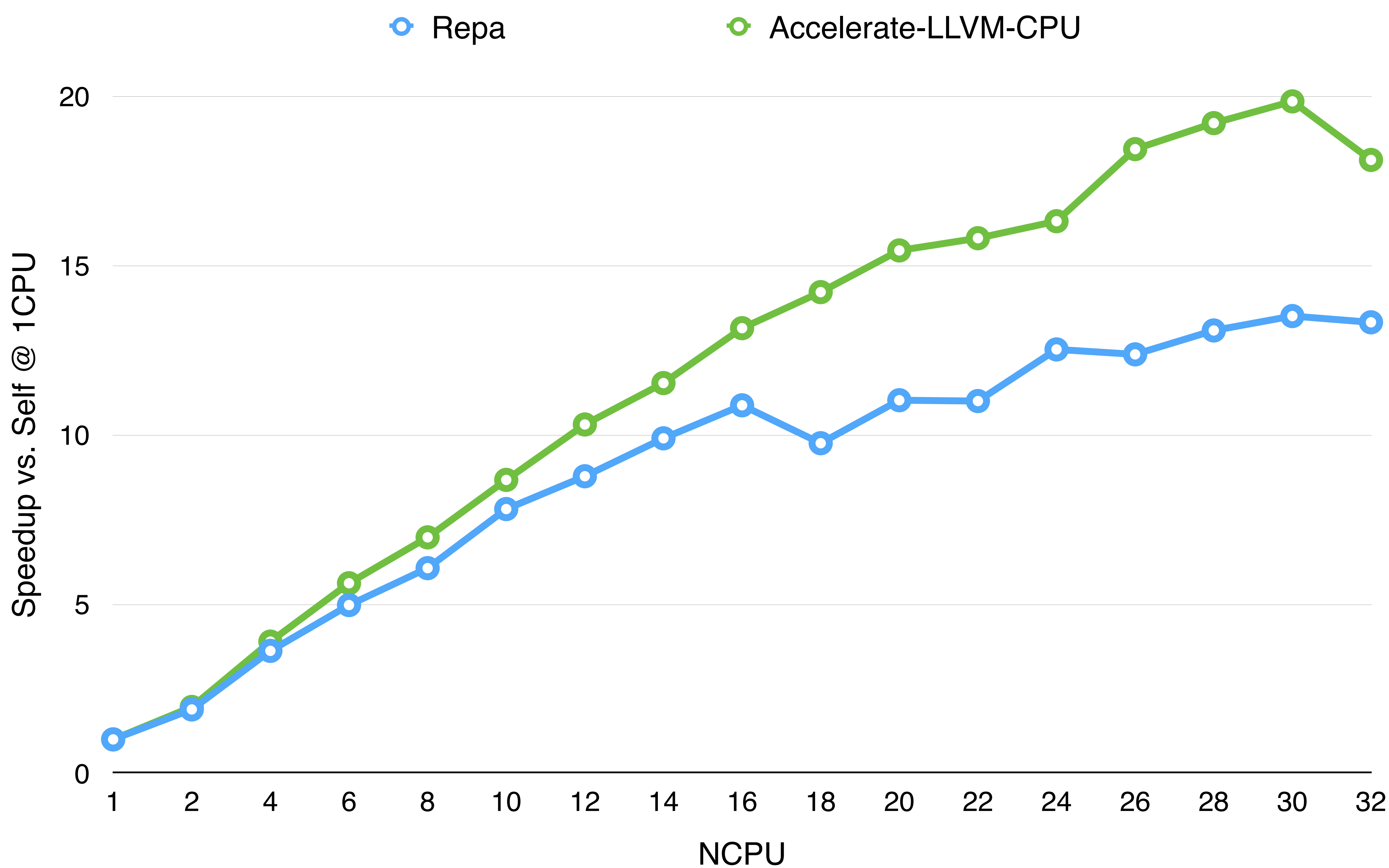
GPU memory management

- Pure functional programs tend to have high allocation/deallocation rates
  - No in-place updates
  - Allocations and deallocations are expensive in CUDA
- Instead if immediately deallocating arrays, keep it for later reuse in the nursery
  - A map from byte size to memory areas of that size
  - Allocate in pages, check the nursery first before allocating fresh data

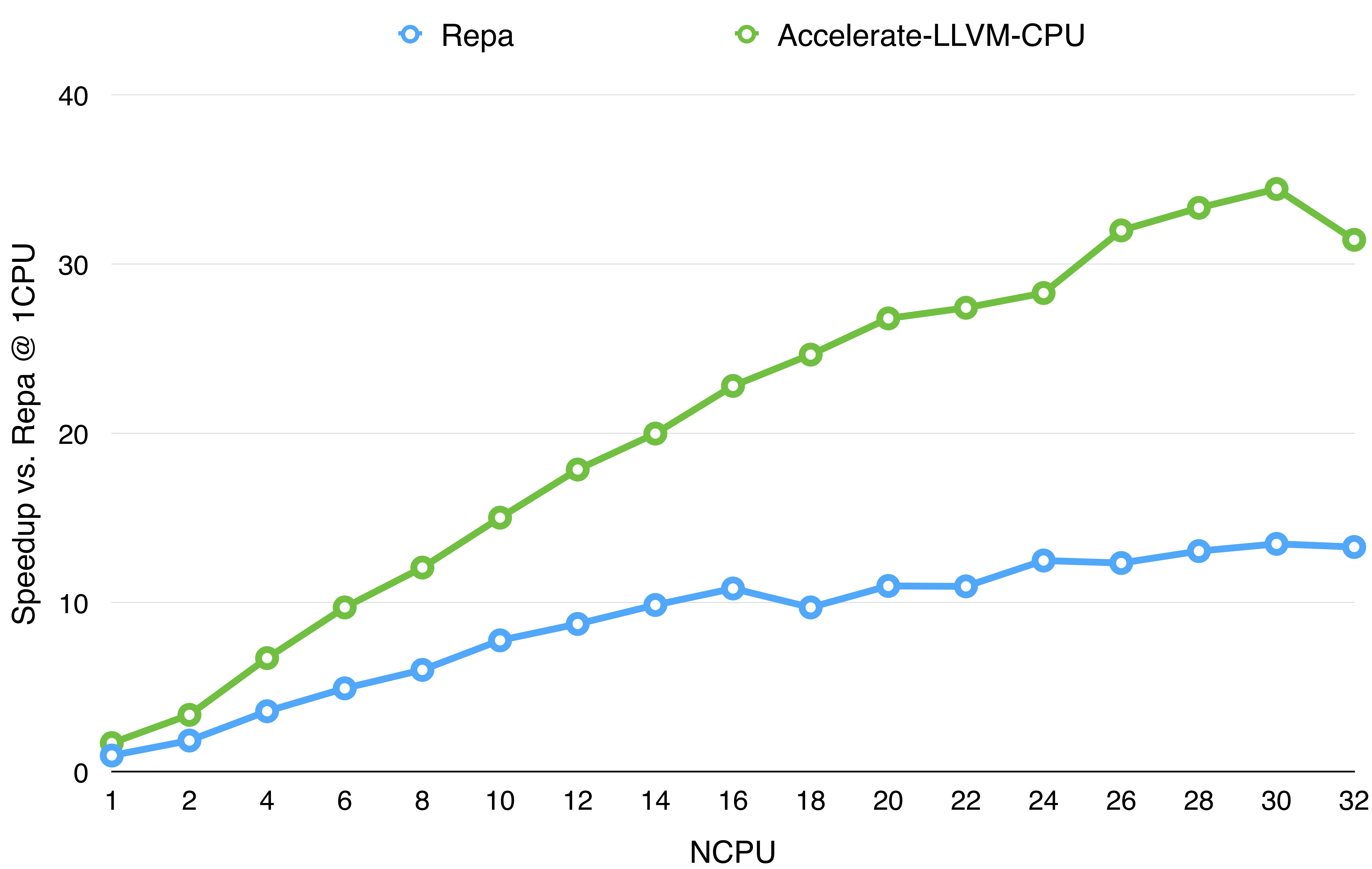
```
type NRS a       = MVar ( IntMap (Seq a) )
data Nursery a   = Nursery (NRS a) (Weak (NRS a))
```

Results

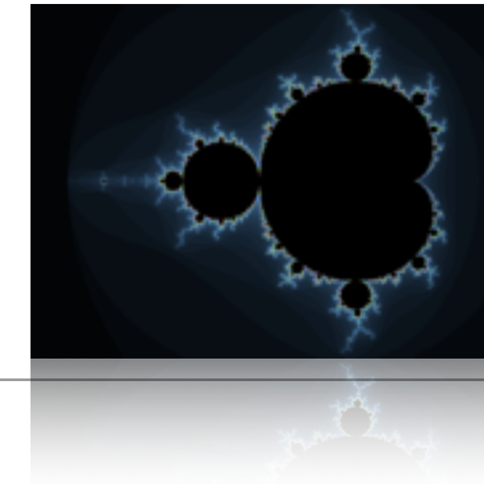
Black-Scholes options pricing

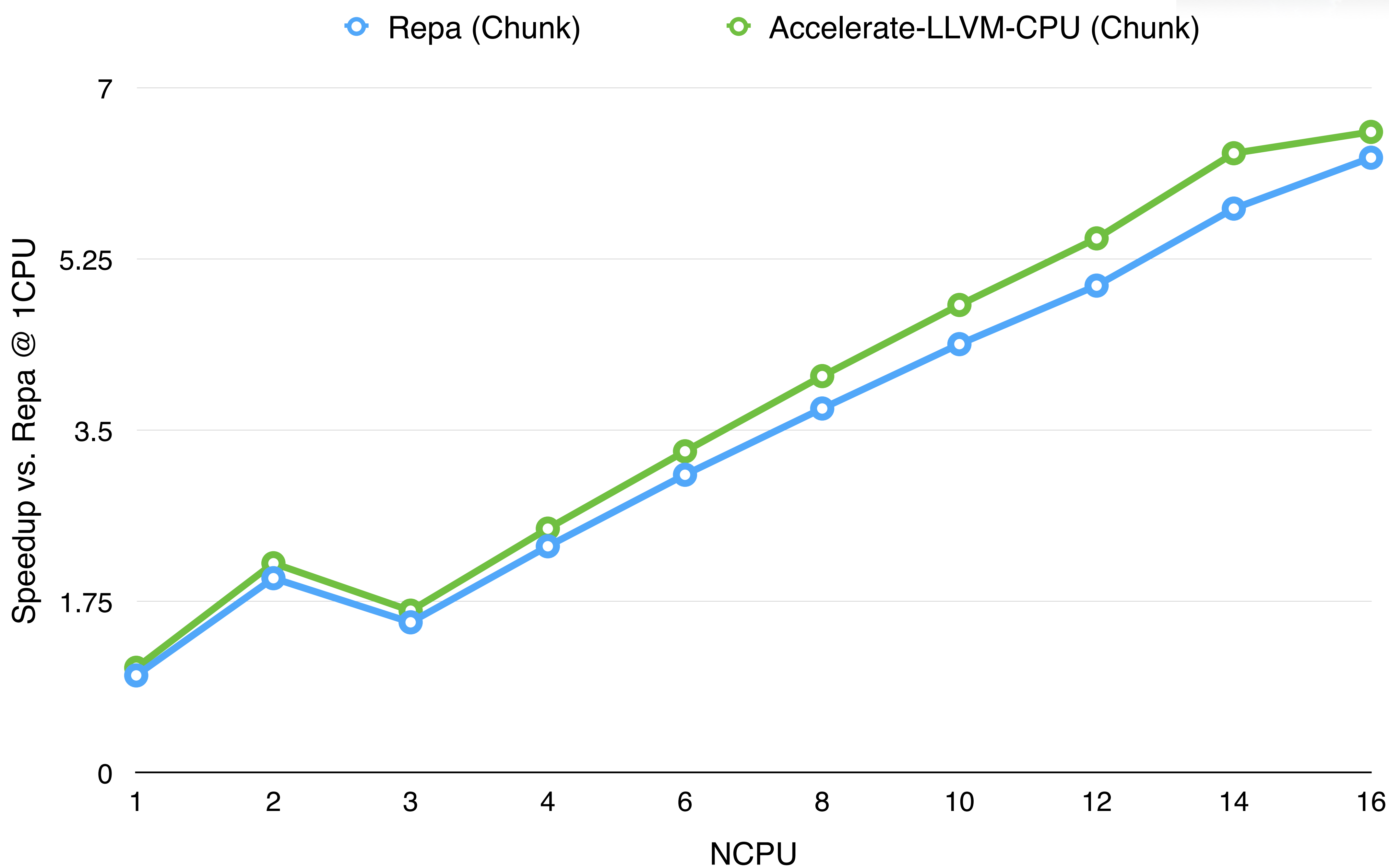


Black-Scholes options pricing

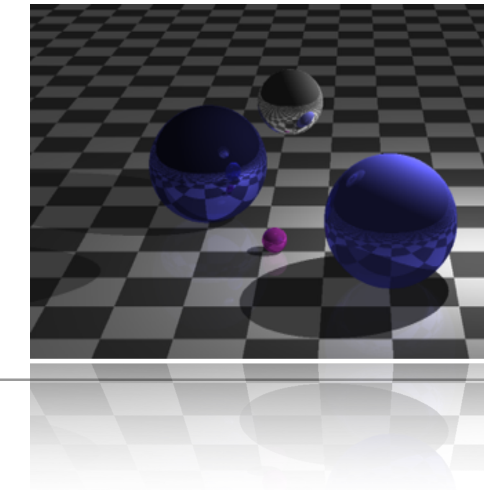


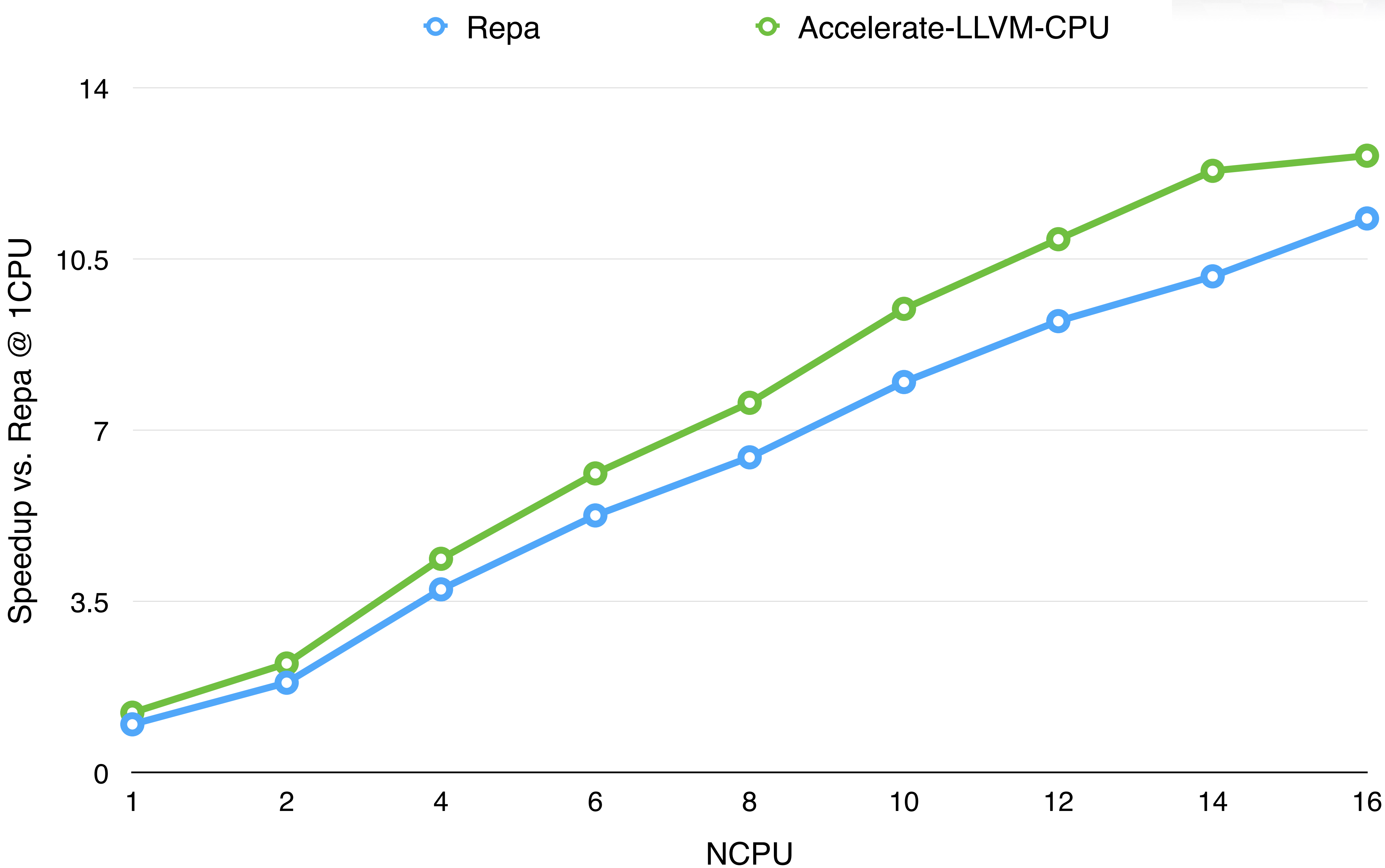
Mandelbrot fractal





Ray tracer

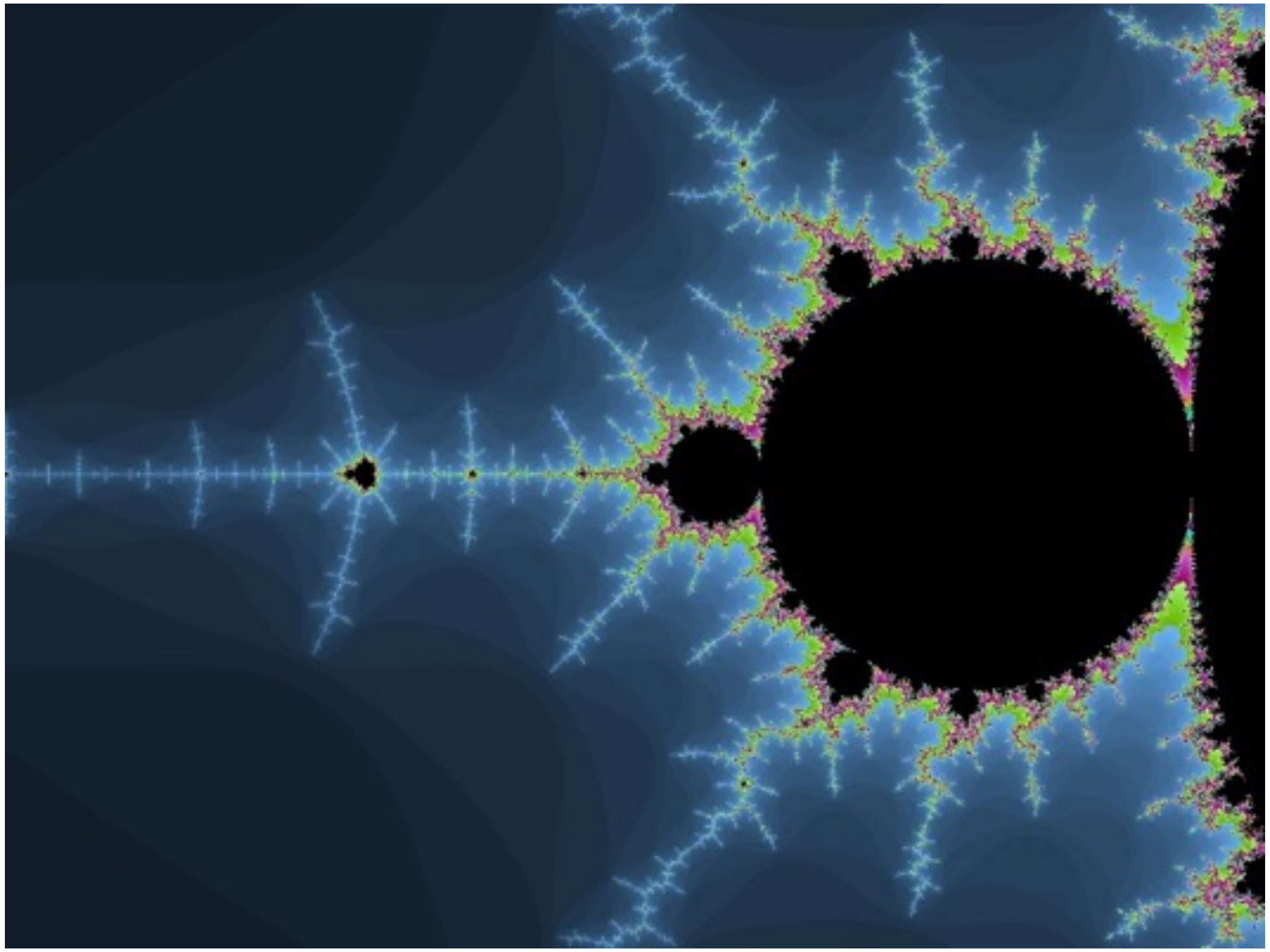




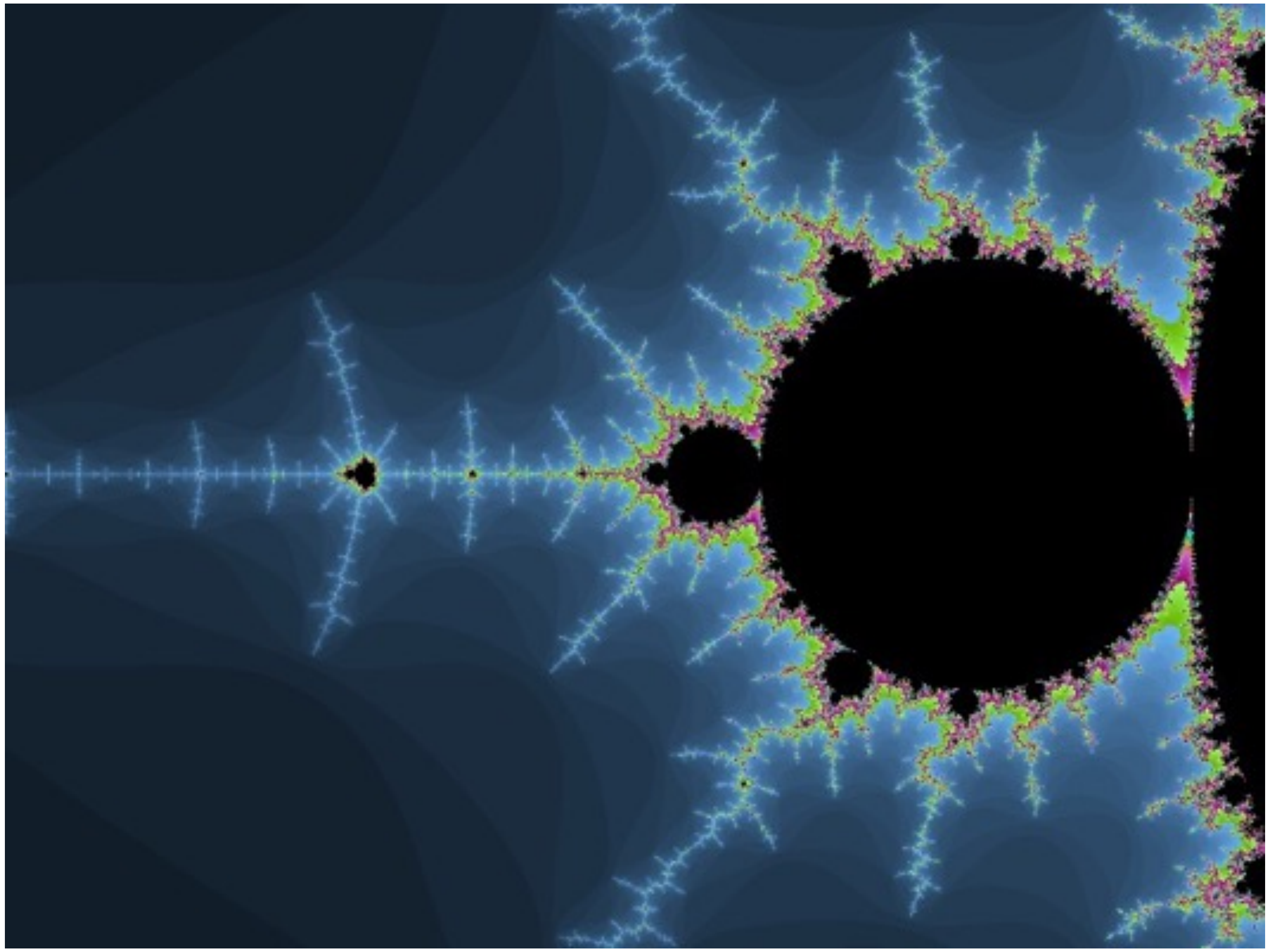
Repa
Accelerate-LLVM-CPU
Speedup vs. Repa @ 1CPU
14
10.5
7
3.5
0
1
2
4
6
8
10
12
14
16
NCPU

Composable scheduling

Unbalanced workloads



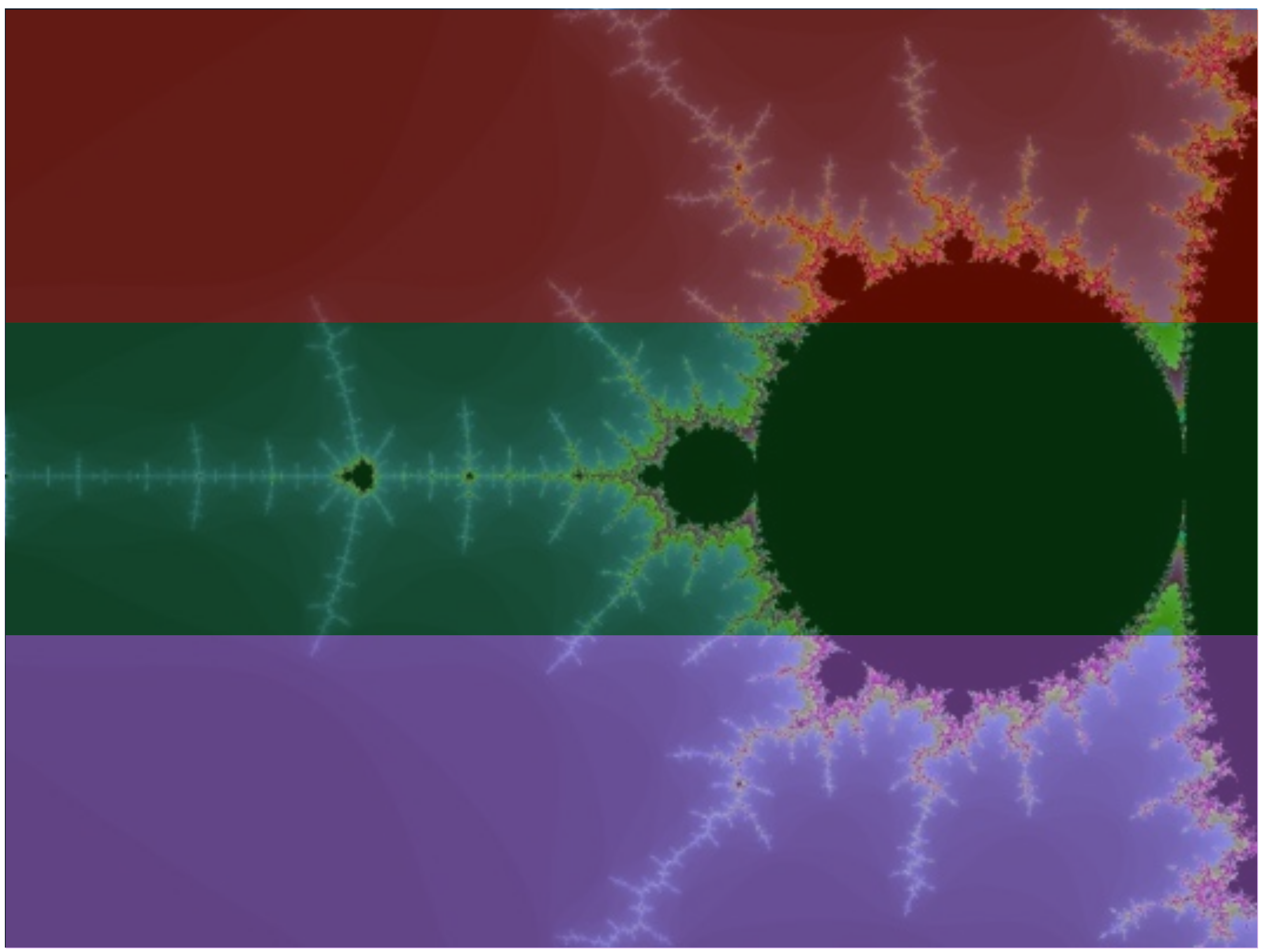
Unbalanced workloads

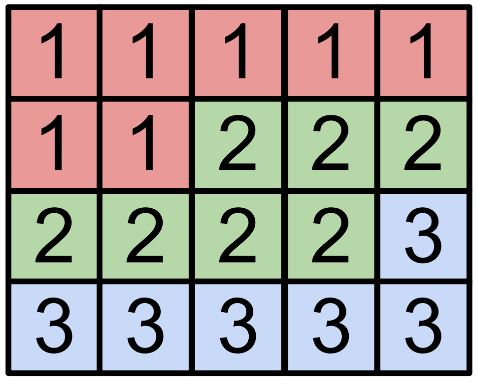


1	1	1	1	1
1	1	2	2	2
2	2	2	2	3
3	3	3	3	3

Chunked

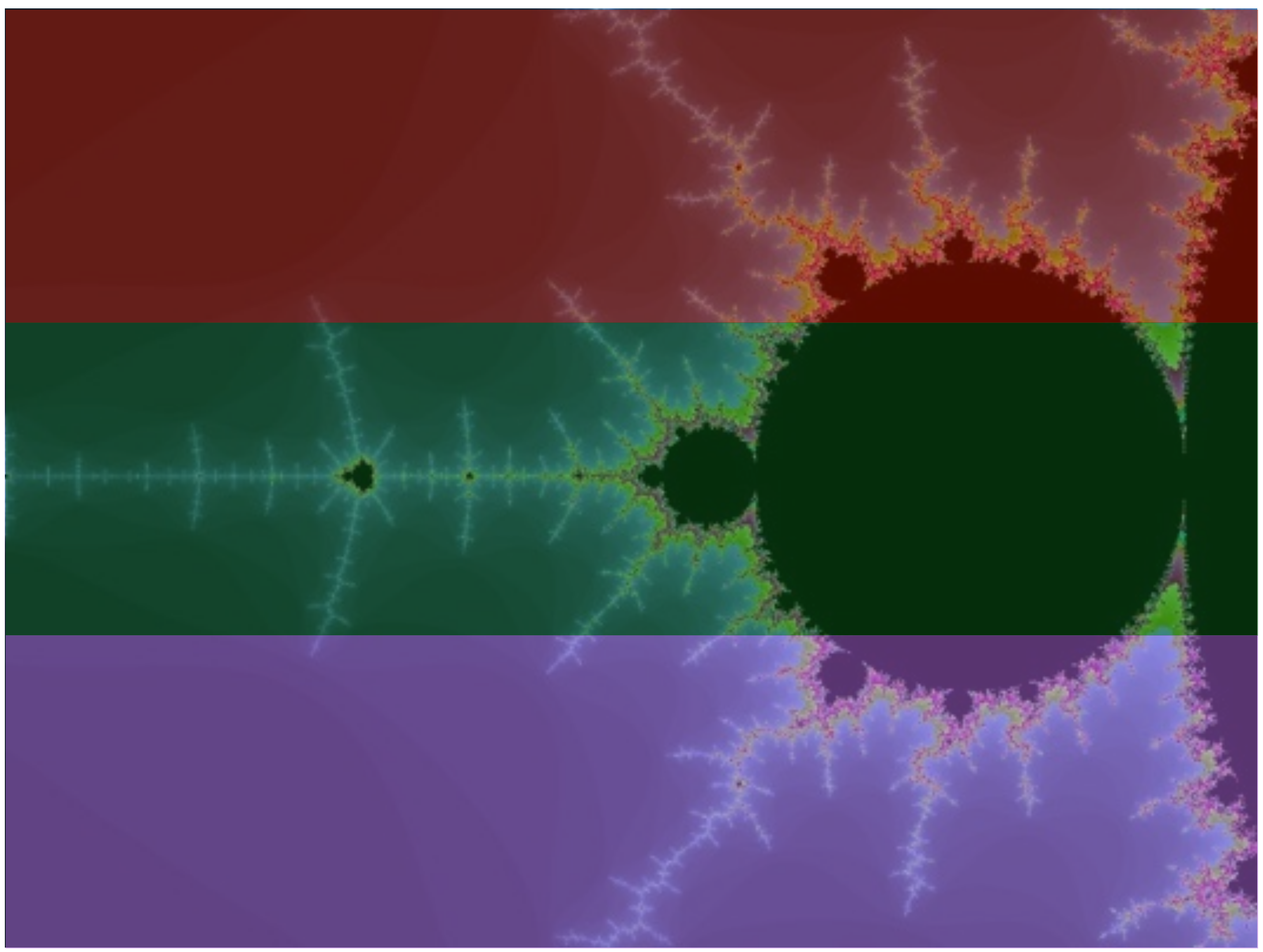
Unbalanced workloads





Chunked

Unbalanced workloads



1	1	1	1	1
1	1	2	2	2
2	2	2	2	3
3	3	3	3	3

Chunked

1	2	3	1	2
3	1	2	3	1
2	3	1	2	3
1	2	3	1	2

Interleaved

Scheduling

- Have: parallel code that performs well
- Want: for that performance to be preserved under composition
  - Unbalanced workloads
  - Non-CPU resources such as the GPU are contending for attention
  - Resources (potentially) have different memory spaces
  - Multiple schedulers need to coordinate effectively
  - Avoid oversubscription (which famously troubles OpenMP)

Work stealing

- For example, a work stealing scheduler might look like:

    1. Steal from local CPUs; *else*
    2. Steal back from the GPU; *else*
    3. Steal from the network; *else*
    4. Sleep to avoid spamming the scheduler; *then* goto step 1

Work stealing

- For example, a work stealing scheduler might look like:

    1. Steal from local CPUs; *else*
    2. Steal back from the GPU; *else*
    3. Steal from the network; *else*
    4. Sleep to avoid spamming the scheduler; *then* goto step 1

- Rather than committing to a particular scheduling algorithm, construct the scheduler — possibly *at runtime* — from reusable components.

Lazy binary splitting

- A resource transformer that provides adaptive work-stealing
  - Unlike eager binary splitting, no manual tuning parameter (TBB, Cilk)
  - Avoids oversubscription
  - Threads use their local deque as an approximation of system load

Lazy binary splitting

- A resource transformer that provides adaptive work-stealing
  - Unlike eager binary splitting, no manual tuning parameter (TBB, Cilk)
  - Avoids oversubscription
  - Threads use their local deque as an approximation of system load
- When the `WorkSearch` returns a unit of work, take the first *ppt* elements and decide what to do with the rest…

Lazy binary splitting

- A resource transformer that provides adaptive work-stealing
  - Unlike eager binary splitting, no manual tuning parameter (TBB, Cilk)
  - Avoids oversubscription
  - Threads use their local deque as an approximation of system load

- When the `WorkSearch` returns a unit of work, take the first *ppt* elements and decide what to do with the rest…

  1. If it is smaller than *ppt* elements, push it back onto the deque

Lazy binary splitting

- A resource transformer that provides adaptive work-stealing
  - Unlike eager binary splitting, no manual tuning parameter (TBB, Cilk)
  - Avoids oversubscription
  - Threads use their local deque as an approximation of system load

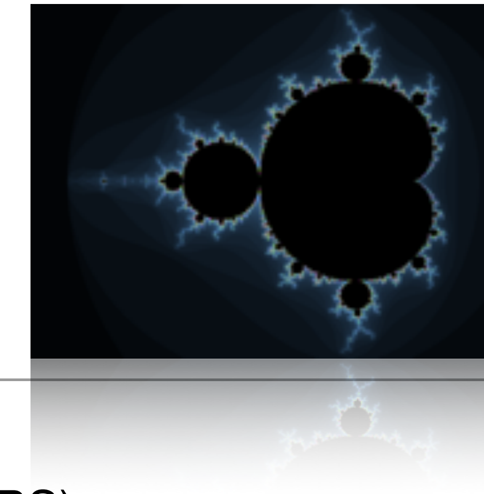
- When the `WorkSearch` returns a unit of work, take the first *ppt* elements and decide what to do with the rest…
  1. If it is smaller than *ppt* elements, push it back onto the deque
  2. If the deque is not empty, push it back

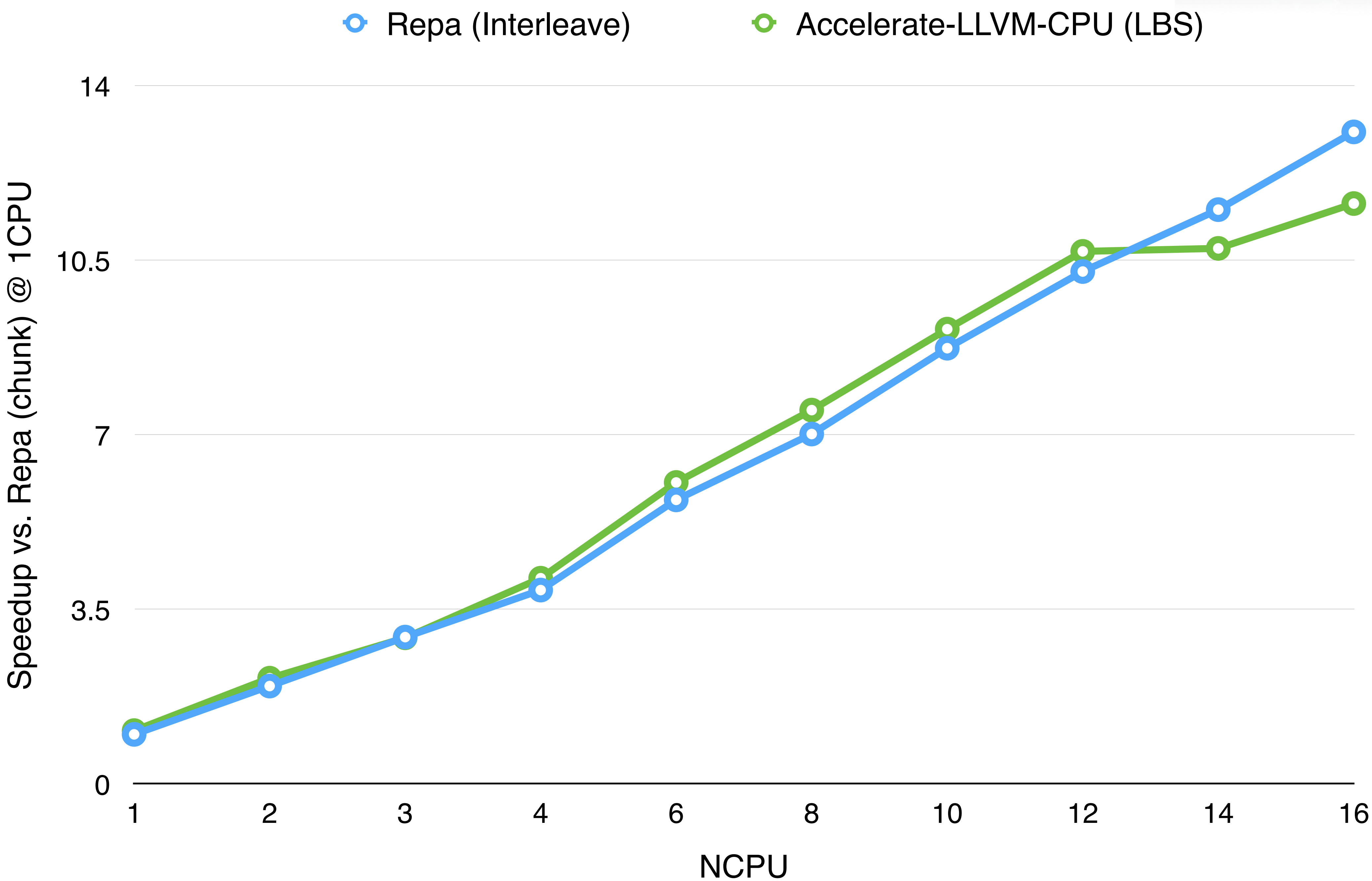
Lazy binary splitting

- A resource transformer that provides adaptive work-stealing
  - Unlike eager binary splitting, no manual tuning parameter (TBB, Cilk)
  - Avoids oversubscription
  - Threads use their local deque as an approximation of system load

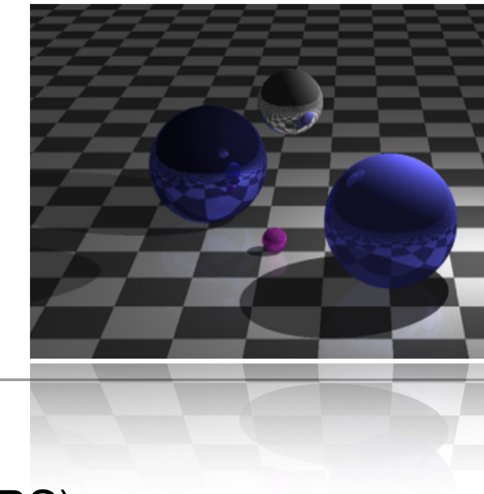
- When the `WorkSearch` returns a unit of work, take the first *ppt* elements and decide what to do with the rest…
  1. If it is smaller than *ppt* elements, push it back onto the deque
  2. If the deque is not empty, push it back
  3. The deque is empty: split it in half and push both pieces back

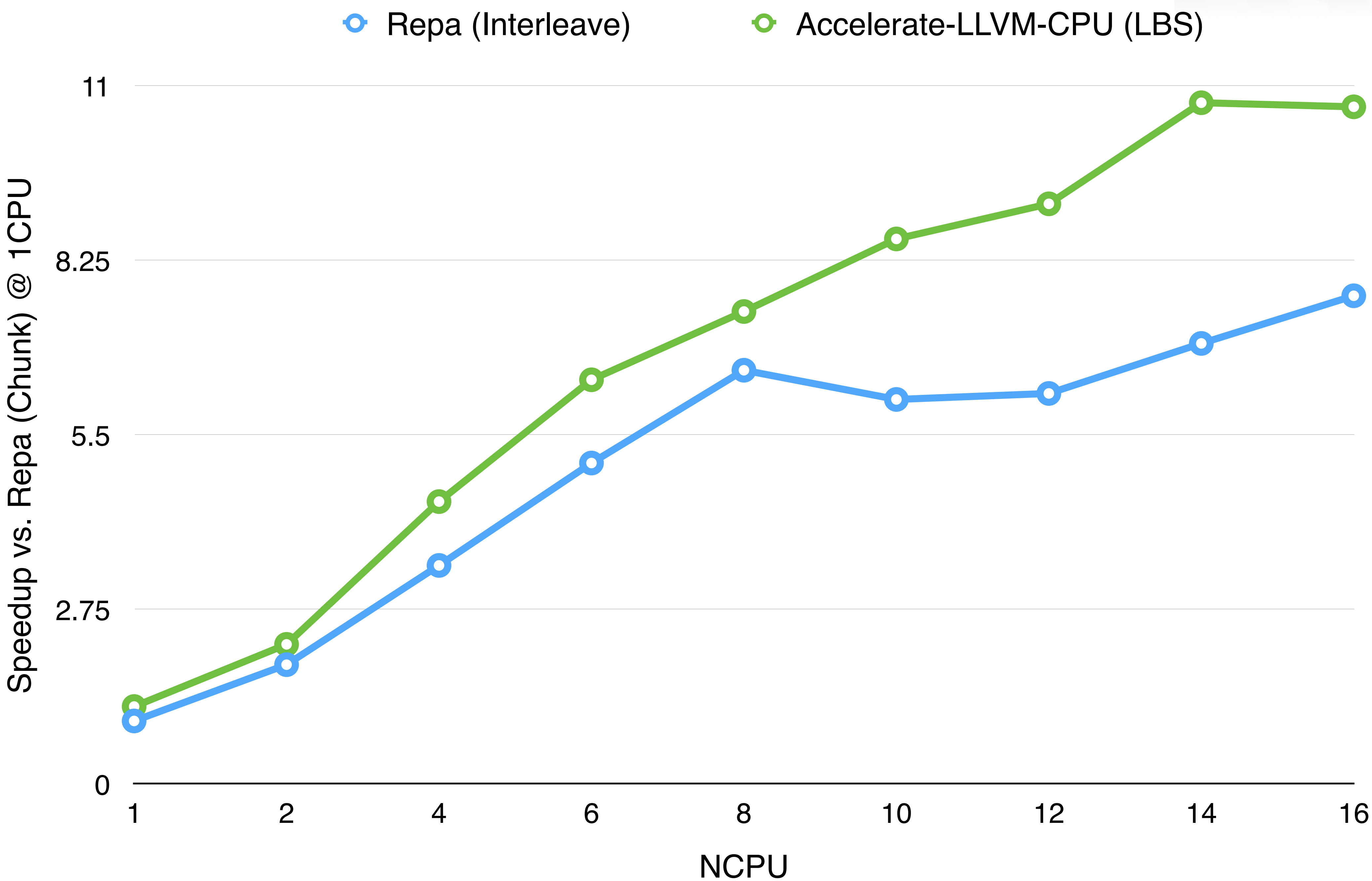
Mandelbrot fractal





Ray tracer





Repa (Interleave)
Accelerate-LLVM-CPU (LBS)
11
8.25
5.5
2.75
0
Speedup vs. Repa (Chunk) @ 1CPU
1
2
4
6
8
10
12
14
16
NCPU

Hybrid CPU/GPU execution

Hybrid CPU/GPU backend

- Goal: compose these new CPU and GPU targeting backings so that the composition evaluates expressions cooperatively
  - Since operations are parameterised by the type of the backend target, this enables easy vertical composition

Hybrid CPU/GPU backend

- Goal: compose these new CPU and GPU targeting backings so that the composition evaluates expressions cooperatively
  - Since operations are parameterised by the type of the backend target, this enables easy vertical composition

```
compileForMulti acc aenv =
  MultiR <$> compileForTarget acc aenv `with` ptxTarget
         <*> compileForTarget acc aenv `with` nativeTarget
```

Hybrid CPU/GPU backend

- Goal: compose these new CPU and GPU targeting backings so that the composition evaluates expressions cooperatively
  - Since operations are parameterised by the type of the backend target, this enables easy vertical composition

```
compileForMulti acc aenv =
  MultiR <$> compileForTarget acc aenv `with` ptxTarget
         <*> compileForTarget acc aenv `with` nativeTarget
```

Class function, implemented
by each backend target

Hybrid CPU/GPU backend

- Goal: compose these new CPU and GPU targeting backings so that the composition evaluates expressions cooperatively
    - Since operations are parameterised by the type of the backend target, this enables easy vertical composition

select specific target

```
compileForMulti acc aenv =
  MultiR <$> compileForTarget acc aenv `with` ptxTarget
         <*> compileForTarget acc aenv `with` nativeTarget
```

Class function, implemented
by each backend target

Executing hybrid programs

- Resource stacks provide easy vertical composition of a scheduler
- Executing hybrid programs collectively requires horizontal composition
  - Can’t simply call each individual backend’s execution code, as we did for compilation
  - Each backend executes a different operation — not just splitting work
  - Multi-step operations like fold, scan, require deep coordination…

Executing hybrid programs

- For simple operations where each element is independent…
  - Add a steal action at the bottom of each resource stack: CPU ⟷ GPU
  - Use a proxy thread that selects which target to launch

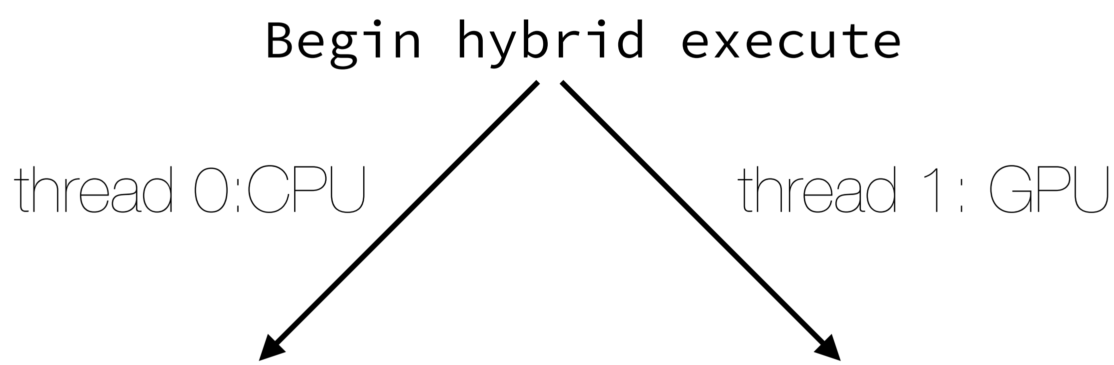
Executing hybrid programs

- For simple operations where each element is independent…
  - Add a steal action at the bottom of each resource stack: CPU ⟷ GPU
  - Use a proxy thread that selects which target to launch

```
Begin hybrid execute
```

Executing hybrid programs

- For simple operations where each element is independent…
  - Add a steal action at the bottom of each resource stack: CPU ⟷ GPU
  - Use a proxy thread that selects which target to launch



Executing hybrid programs

- For simple operations where each element is independent…
  - Add a steal action at the bottom of each resource stack: CPU ⟷ GPU
  - Use a proxy thread that selects which target to launch

```
Begin hybrid execute
```

thread 0:CPU

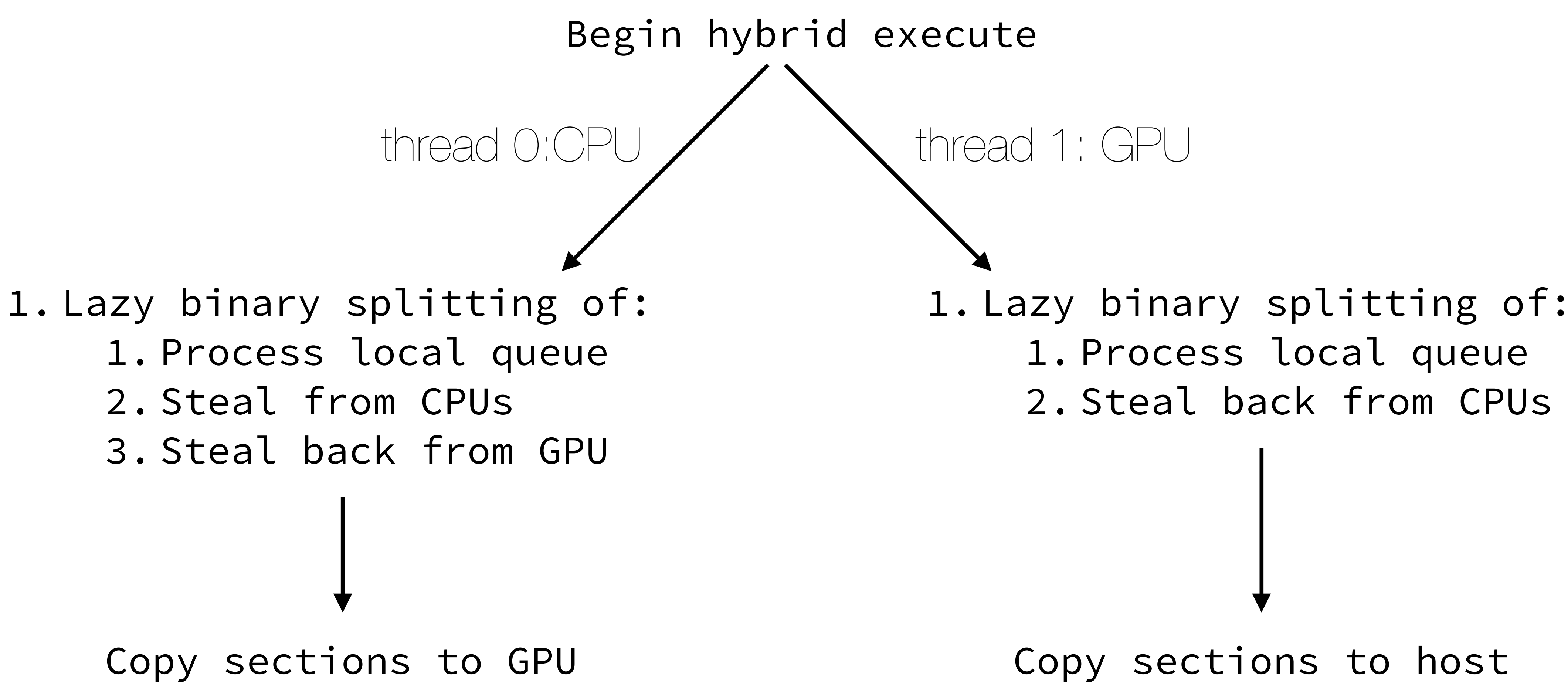
```
1. Lazy binary splitting of:
    1. Process local queue
    2. Steal from CPUs
    3. Steal back from GPU
```

thread 1: GPU

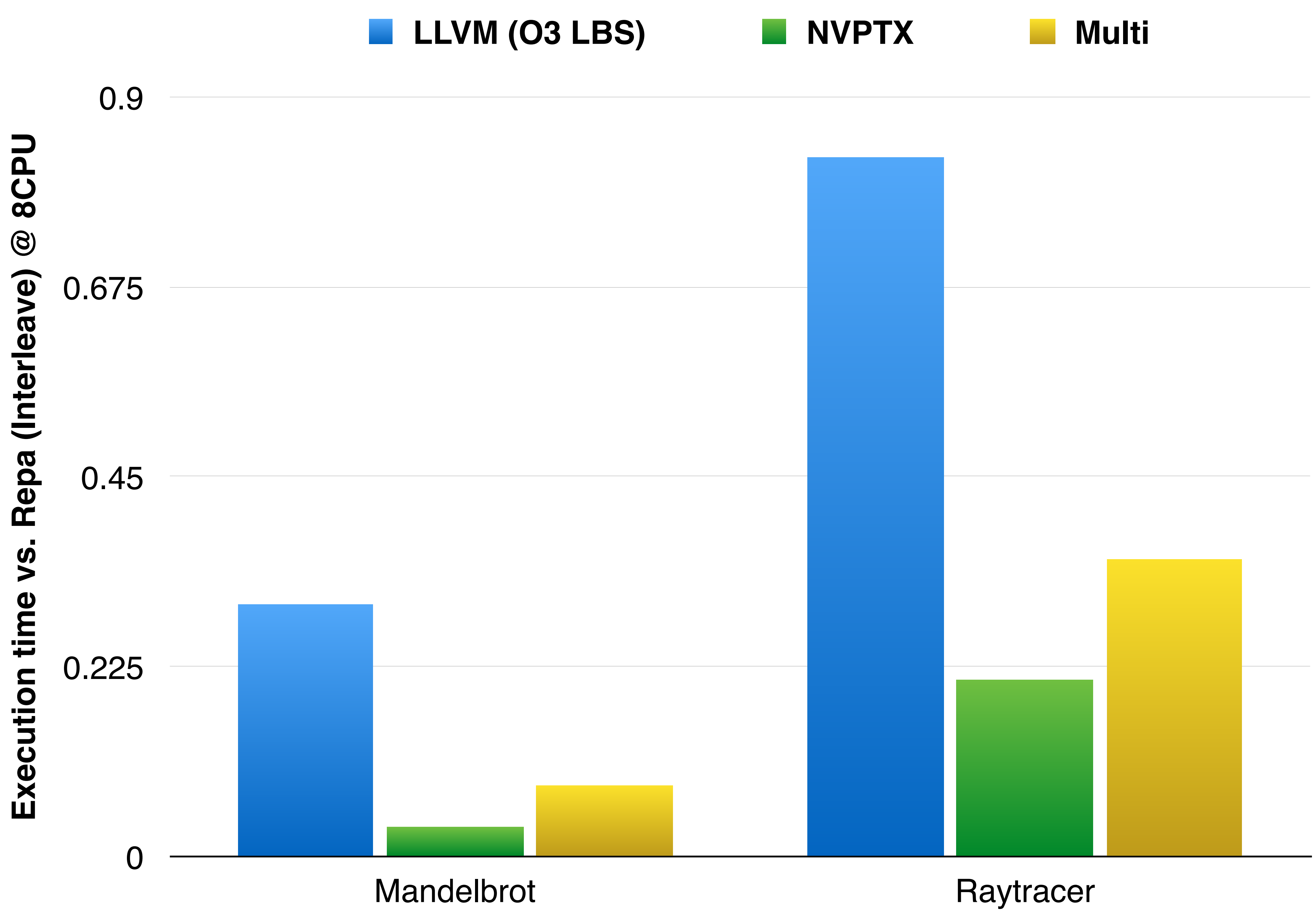
```
1. Lazy binary splitting of:
    1. Process local queue
    2. Steal back from CPUs
```

Executing hybrid programs

- For simple operations where each element is independent…
  - Add a steal action at the bottom of each resource stack: CPU ⟷ GPU
  - Use a proxy thread that selects which target to launch



Results



LLVM (O3 LBS)
NVPTX
Multi
Execution time vs. Repa (Interleave) @ 8CPU
0.9
0.675
0.45
0.225
0
Mandelbrot
Raytracer

Results

- In progress…
  - Since the GPU is much faster than the 8 CPUs (10x) it quickly finishes its work and steals most of the CPU work before the CPUs can contribute
  - Investigate a different initial split (currently 50/50) or steal strategy