TMBL Kernels for CUDA GPUs Compile Faster Using PTX

Birkbeck
UNIVERSITY OF LONDON

Tony E Lewis

George D Magoulas

Two Major Approaches to GPU Acceleration of GP

Data parallel
Compile new GPU code for each new batch

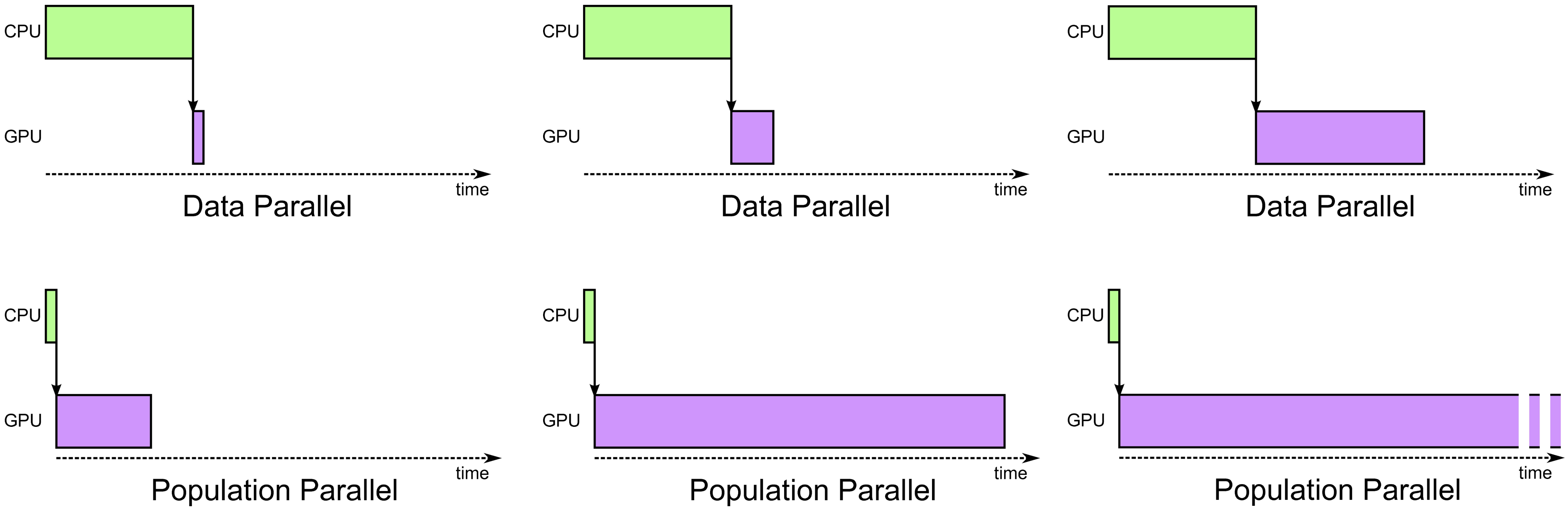
Population parallel
Write one GPU interpreter to process all batches

The Aim of the Work:
To Minimise the Weakness of Data-parallel

Data parallel
Evaluation: *very fast*
Compilation: *long*

Population parallel
Evaluation: *fast*
Compilation: *none*

The Problem: Compilation Stops Small Datasets Getting Top Speed



Two Strategies to Ease Load for Compiler; This Talk is about the First

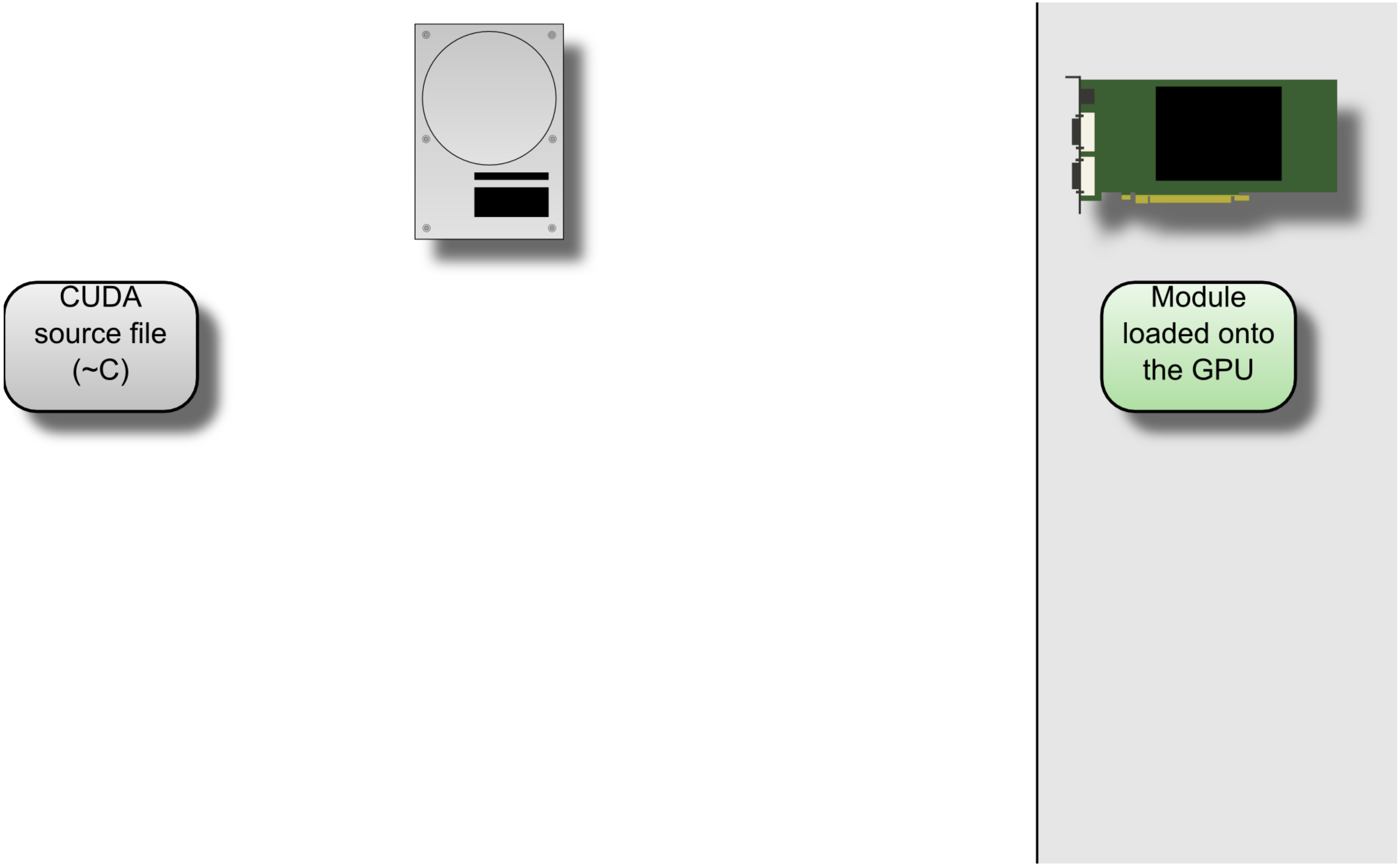
1. PTX

Write the individuals in a lower level language

2. Alignment

Exploit similarities between individuals

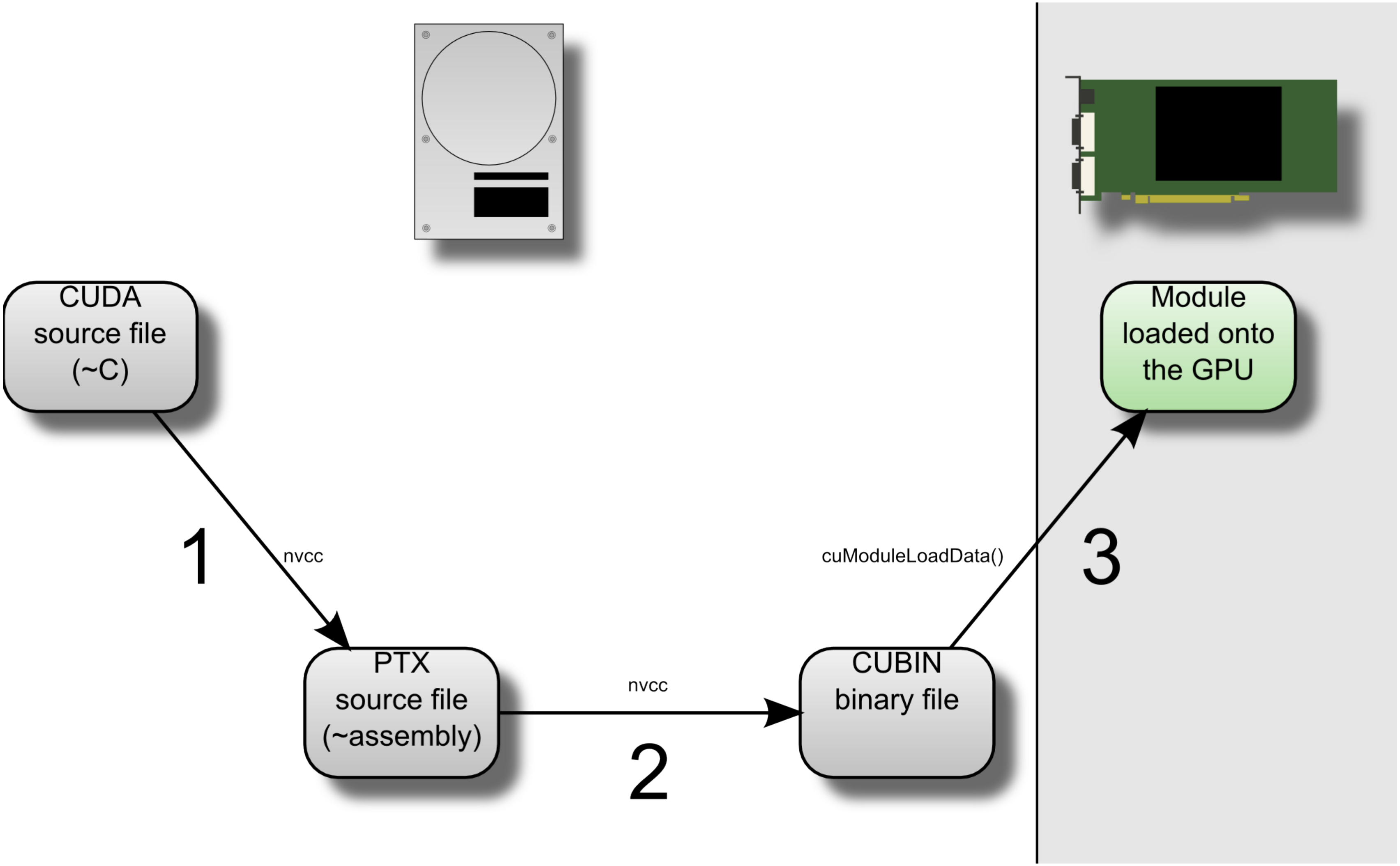
Compilation Creates a GPU-ready Binary from C Source Code



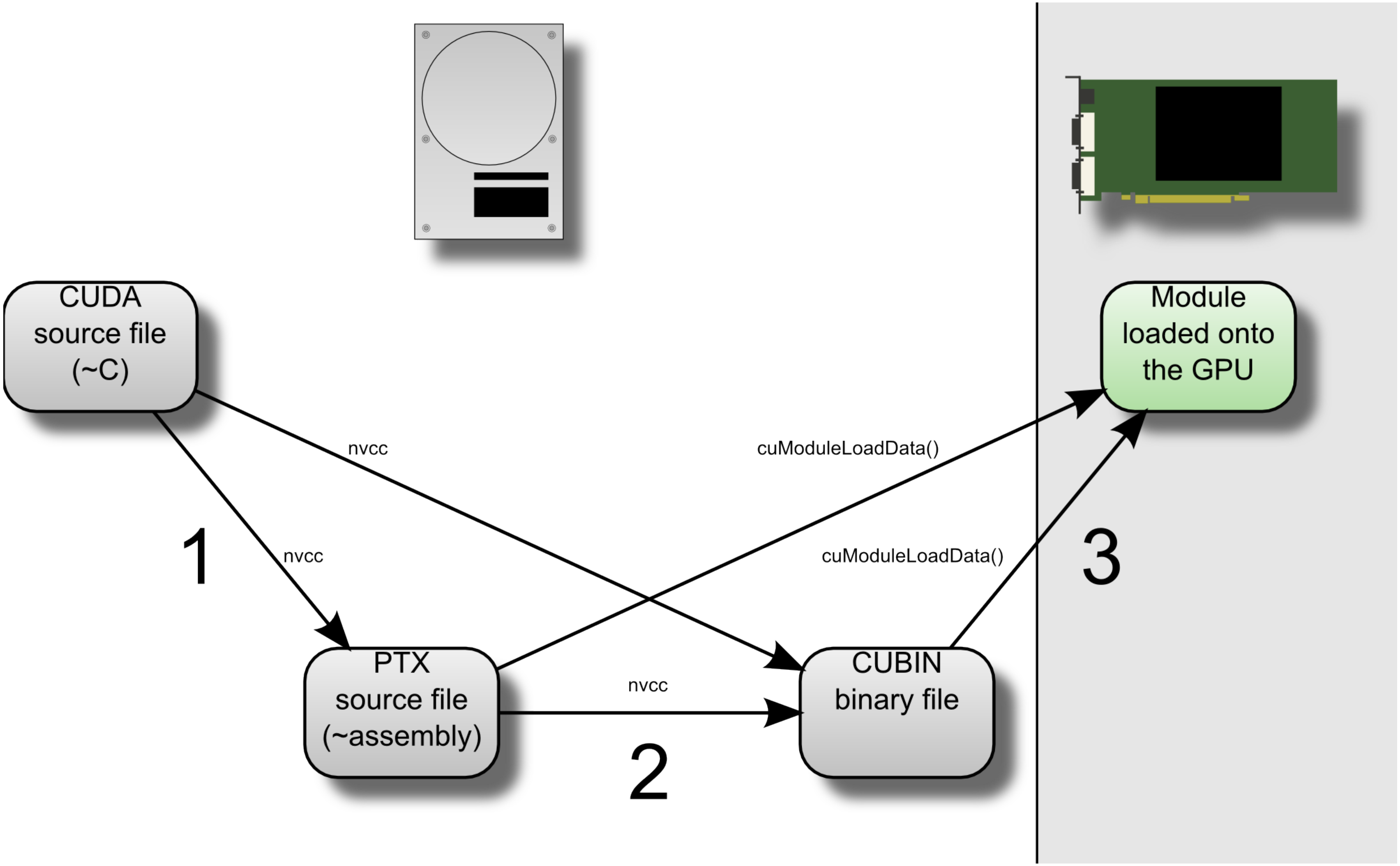
CUDA source file (~C)

Module loaded onto the GPU

Compilation Uses Two Slow Steps; This Work Eliminates the First



Compilation Uses Two Slow Steps; This Work Eliminates the First



PTX is a Bit Like Assembly

C Example

```c
slot0 = -1.64101672f;

slot4 += slot3;

slot1 -= testcase0;

slot0 *= slot3;

slot2 = (
 (slot3 == 0.0f) ?
 0.0f :
 slot2/slot3
);
```

PTX Example

```
mov.f32 %slot0, 0fBFD20CD6;

add.f32 %slot4, %slot4, %slot3;

sub.f32 %slot1, %slot1, %testcase0;

mul.f32 %slot0, %slot0, %slot3;

div.full.f32 %slot2, %slot2, %slot3;
setp.eq.f32 %divPred, %slot3, 0f00000000;
selp.f32 %slot2, 0f00000000, %slot2, %divPred;
```

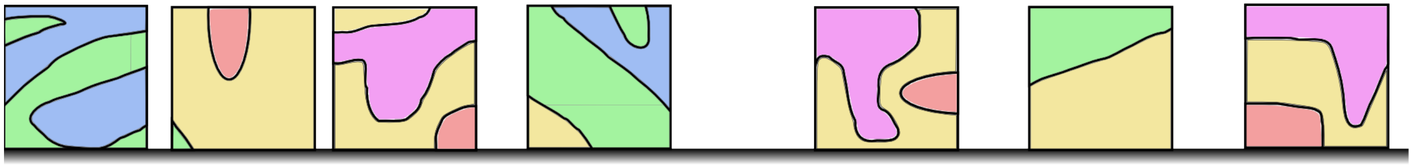
Take a Step Back:
What is the Reason For Doing This Work?

Take a Step Back:
What is the Reason For Doing This Work?

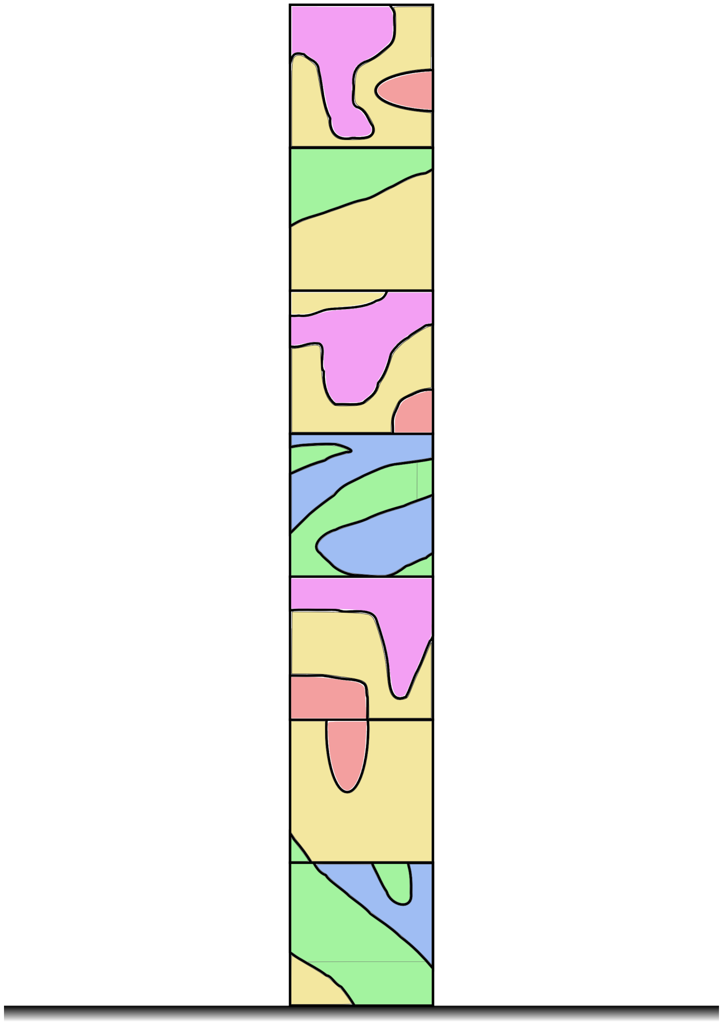
Long Term Fitness Growth

Thought Experiment:

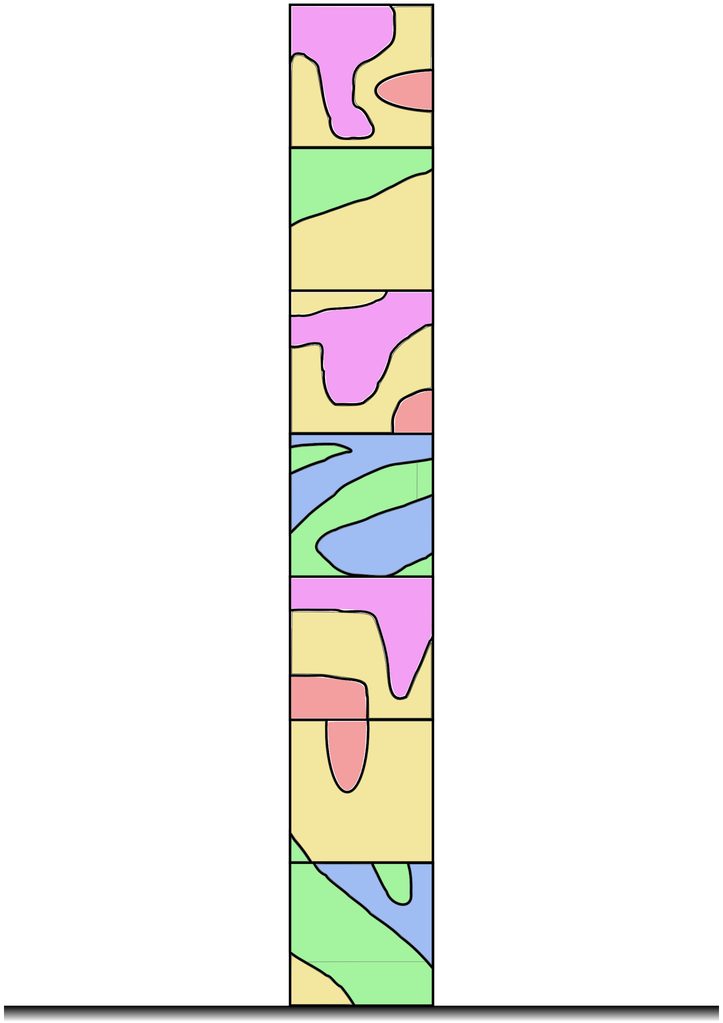
Thought Experiment:
Toy Blocks

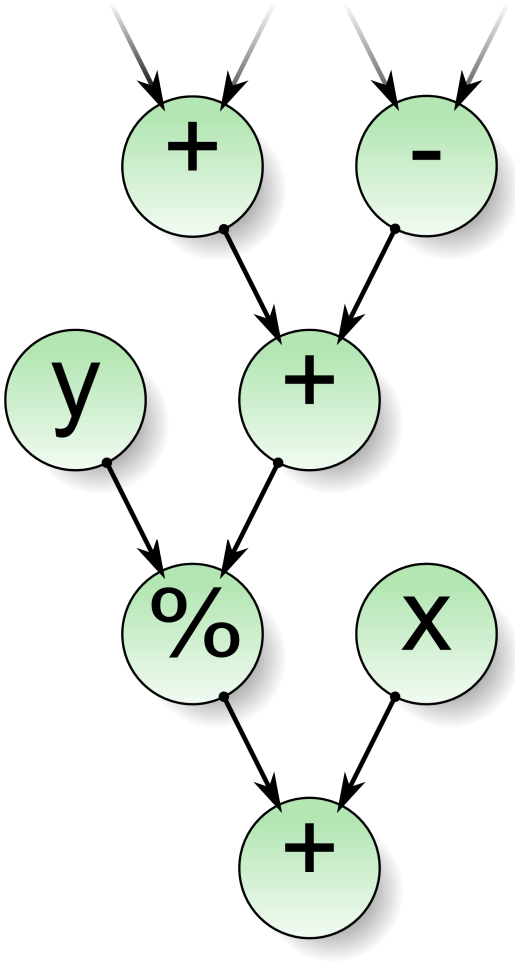


Thought Experiment:
A Tower of Blocks



The Same Problem Is Faced by a GP Tree





How Can We Encourage Long Term Fitness Growth?

How Can We Encourage Long Term Fitness Growth?

Encourage *tweaks:*

Mutations that can easily change behaviour without ruining existing functionality

A Representation to Encourage Tweaks

Linear form not node-based

Registers not stack

Iterated execution not point of execution

Instructions that modify not overwrite

Long programs

The Result: TMBL

Tweaking a Tower of Blocks Leads to a TMBL:
Pursuing Long Term Fitness Growth in Program Evolution
Tony E Lewis,George D Magoulas
2010, IEEE Congress on Evolutionary Computation (CEC)
(pages 4465-4472)

`takesatmbl.wordpress.com`

PTX is a Bit Like Assembly

C Example

```
slot0 = -1.64101672f;

slot4 += slot3;

slot1 -= testcase0;

slot0 *= slot3;

slot2 = (
 (slot3 == 0.0f) ?
 0.0f :
 slot2/slot3
);
```

PTX Example

```
mov.f32 %slot0, 0fBFD20CD6;

add.f32 %slot4, %slot4, %slot3;

sub.f32 %slot1, %slot1, %testcase0;

mul.f32 %slot0, %slot0, %slot3;

div.full.f32 %slot2, %slot2, %slot3;
setp.eq.f32 %divPred, %slot3, 0f00000000;
selp.f32 %slot2, 0f00000000, %slot2, %divPred;
```

...but PTX isn't Exactly Like Assembly

Doesn't directly correspond with resulting binary

Eg. Many registers get compiled to few

Will PTX Code Evaluate Slower?

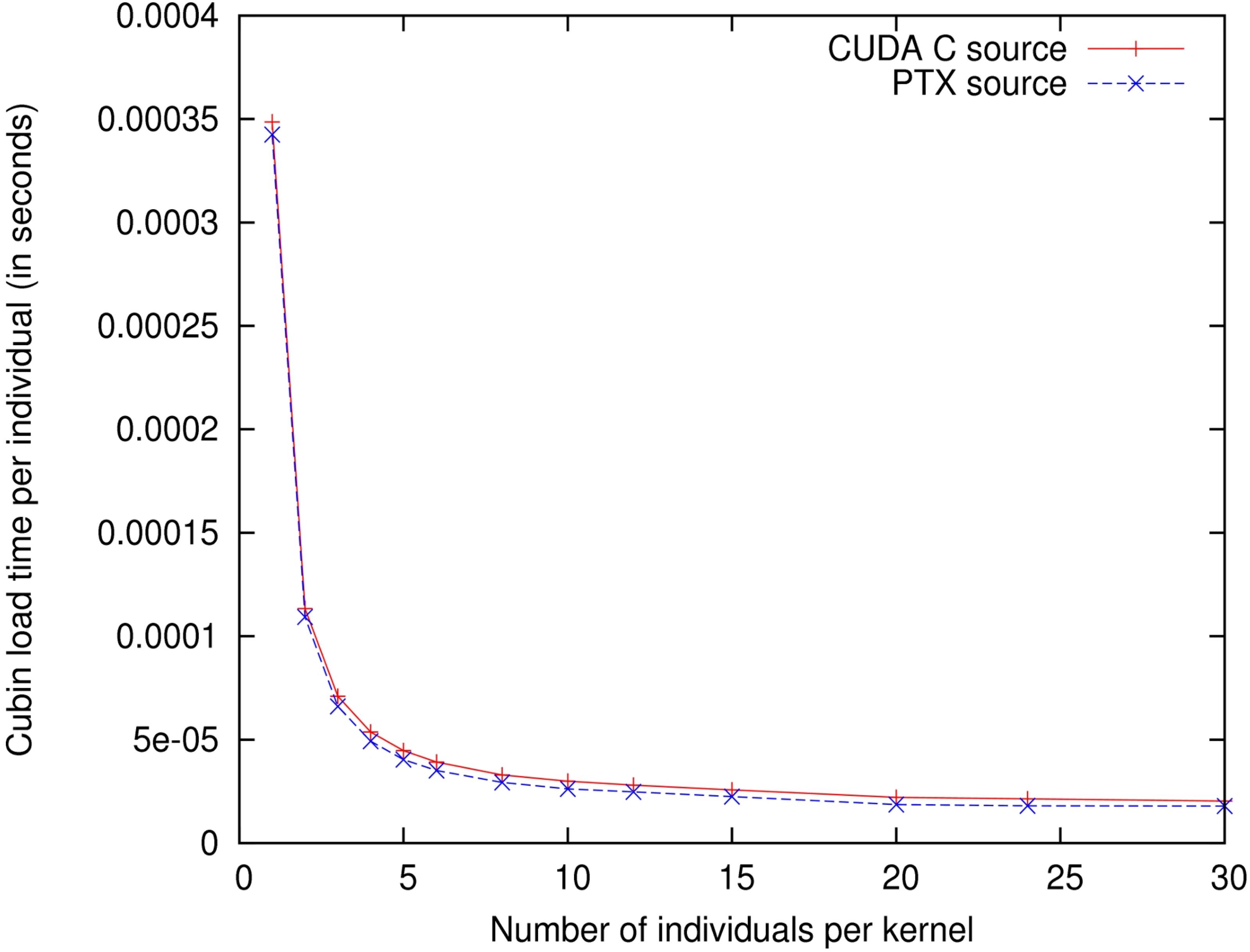
Maybe Yes:

Competing with the CUDA compiler's developers

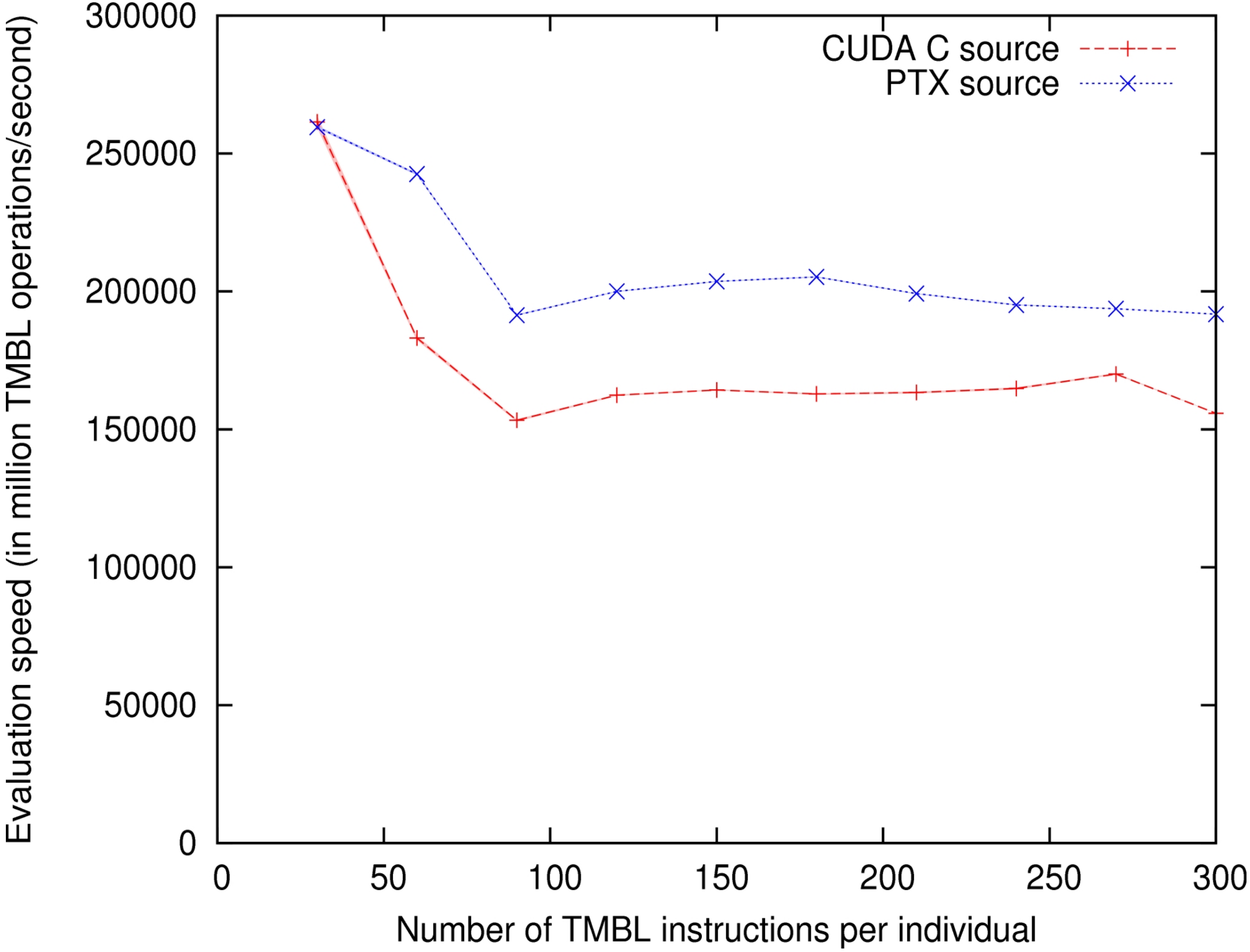
Maybe No:

We know our code better than the compiler does:
Can guarantee non-divergent branches
Can use non-divergent instructions (`a=b?c:d`)

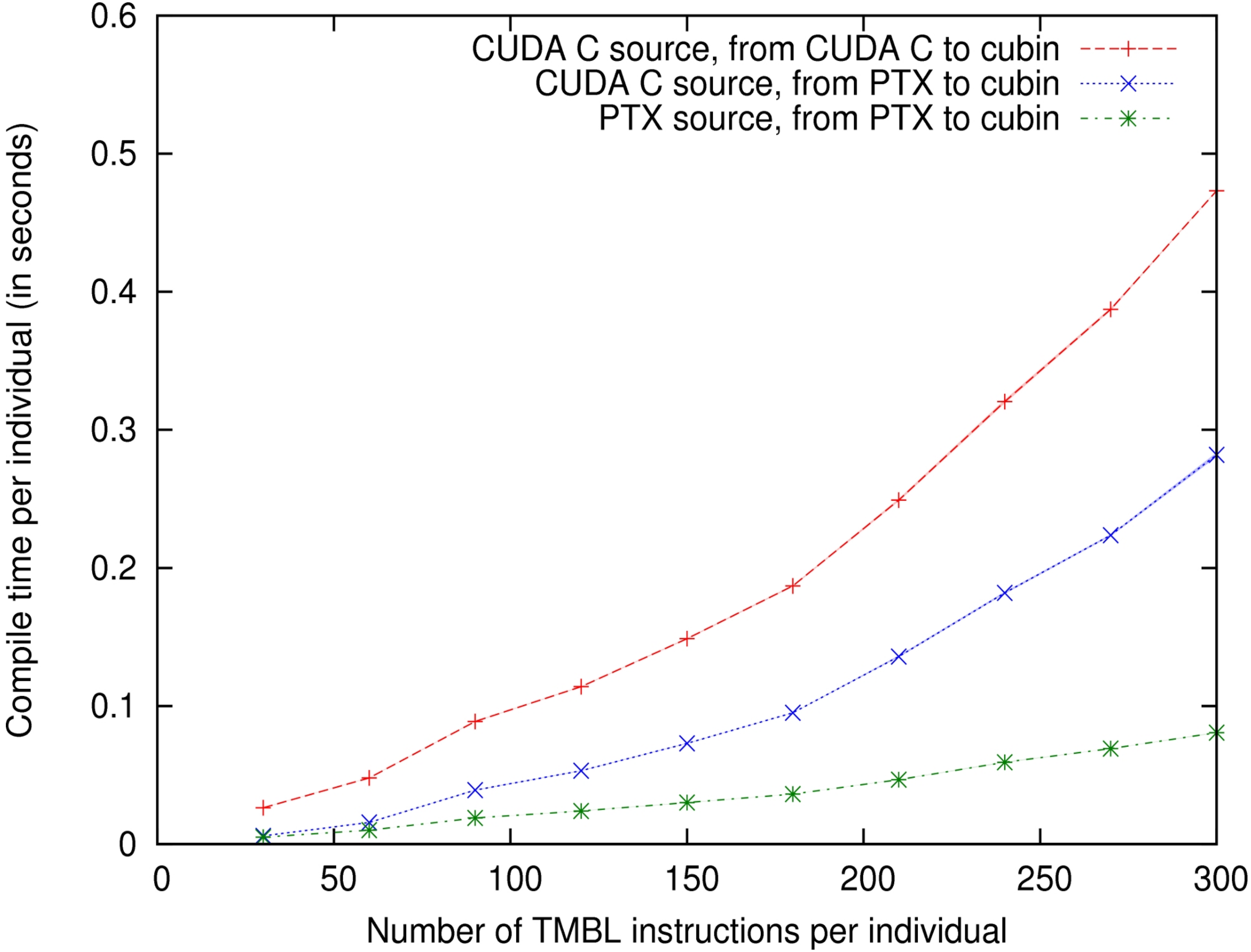
Results:
Load time is small



Results:
Evaluation Speed is Improved



Results:
Compile Time is Considerably Reduced (~5.8x)



Conclusions

Complexity

Maintainability

Effectiveness

Possibility of going further

Thanks

EPSRC

Reviewers

You