Alpinist
an Annotation-Aware GPU Program Optimizer

Ömer Şakar[1] Mohsen Safari[1] Marieke Huisman[1] Anton Wijs[2]

Formal Methods and Tools, University of Twente, Enschede, The Netherlands

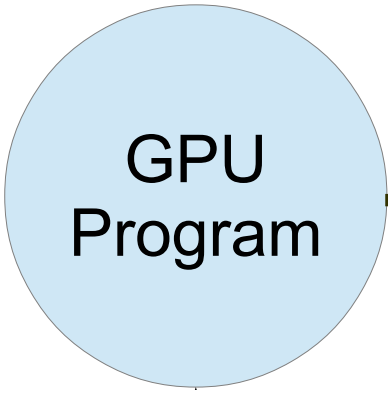
Software Engineering & Technology, Eindhoven University of Technology, Eindhoven, The Netherlands

CHEOPS
VERIFIED CONSTRUCTION OF CORRECT AND OPTIMISED PARALLEL SOFTWARE

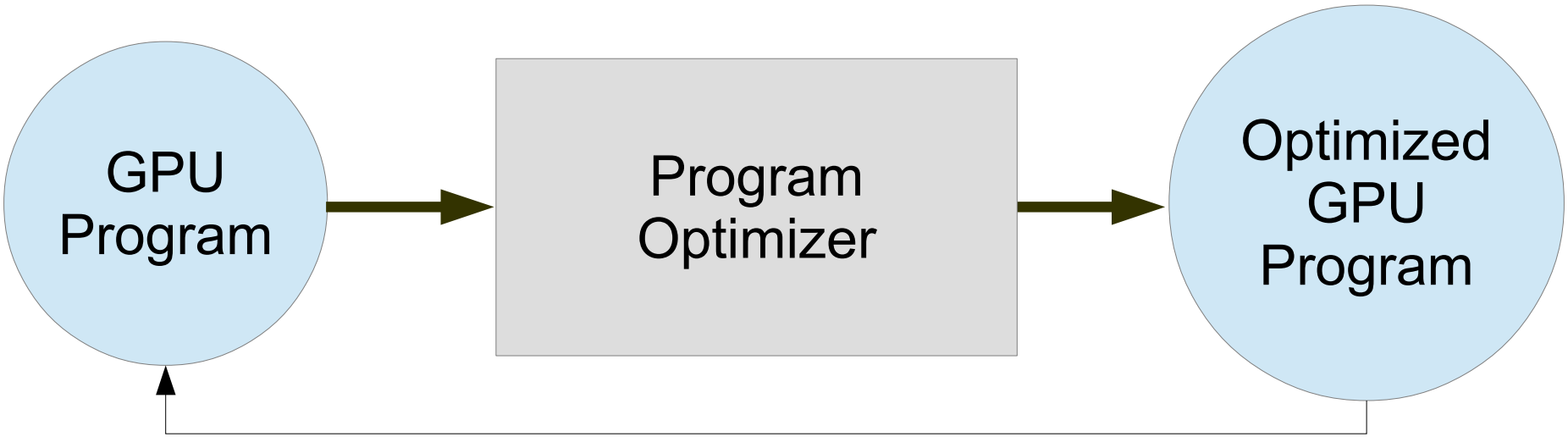
April 7, 2022

Mercedes
MAXIMAL RELIABILITY OF CONCURRENT AND DISTRIBUTED SOFTWARE

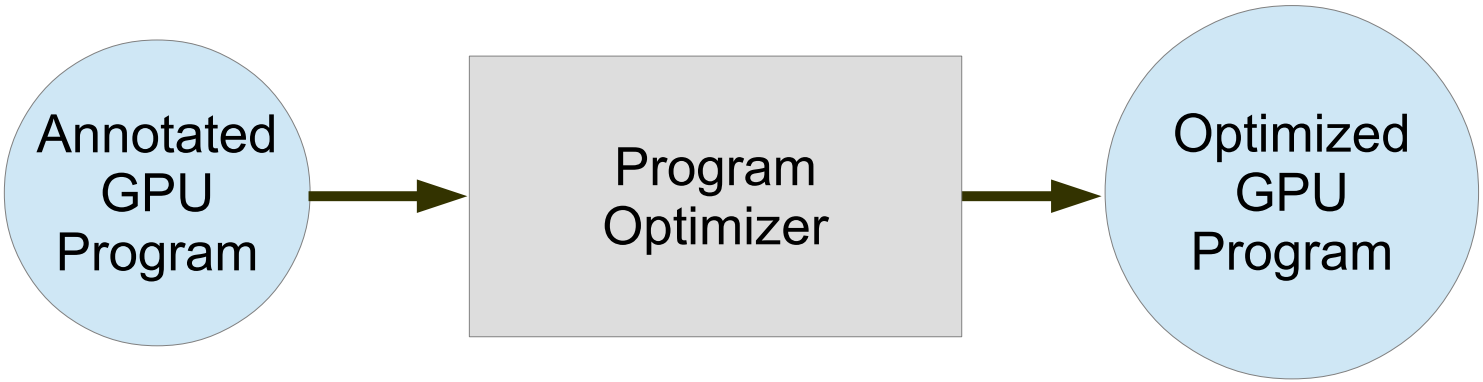
GPU Program Development Cycle



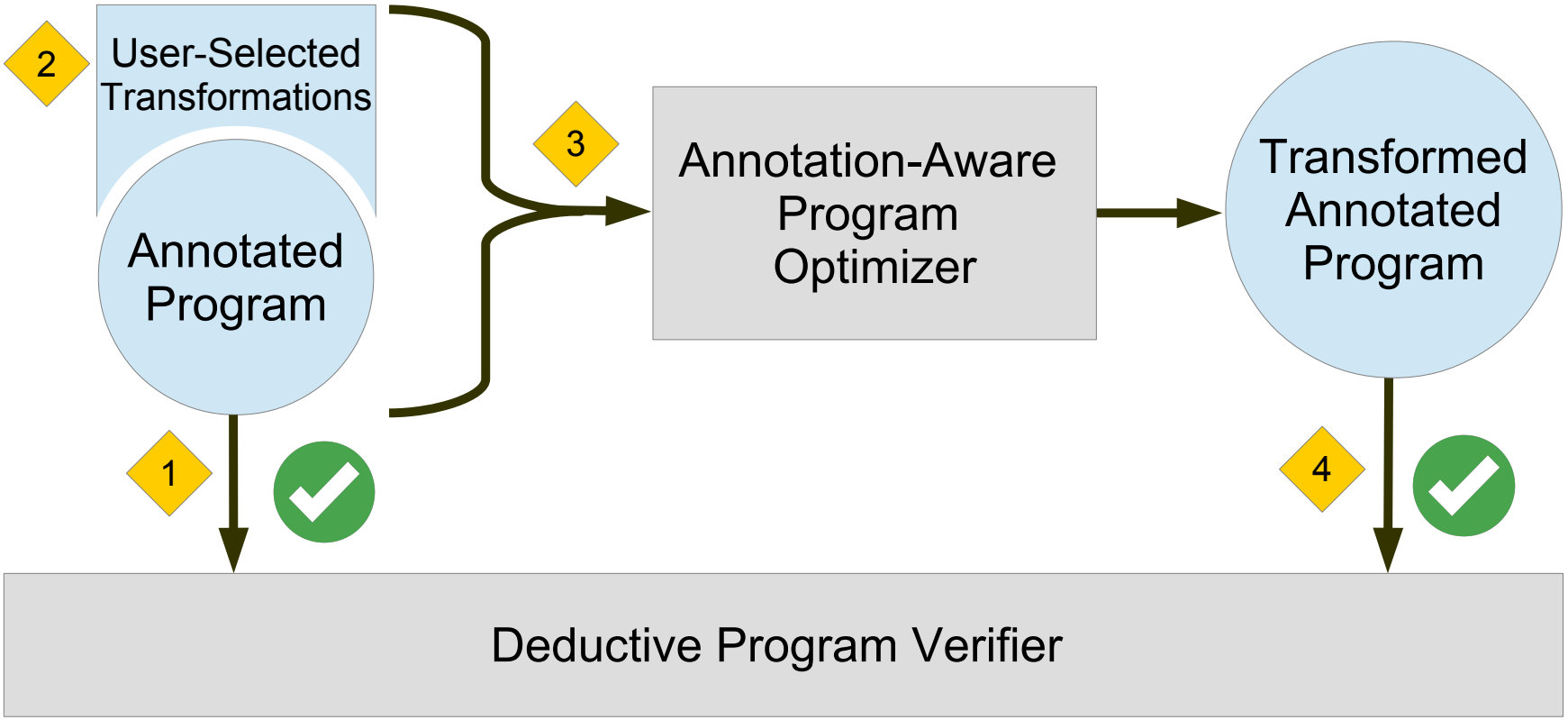
GPU Program Development Cycle



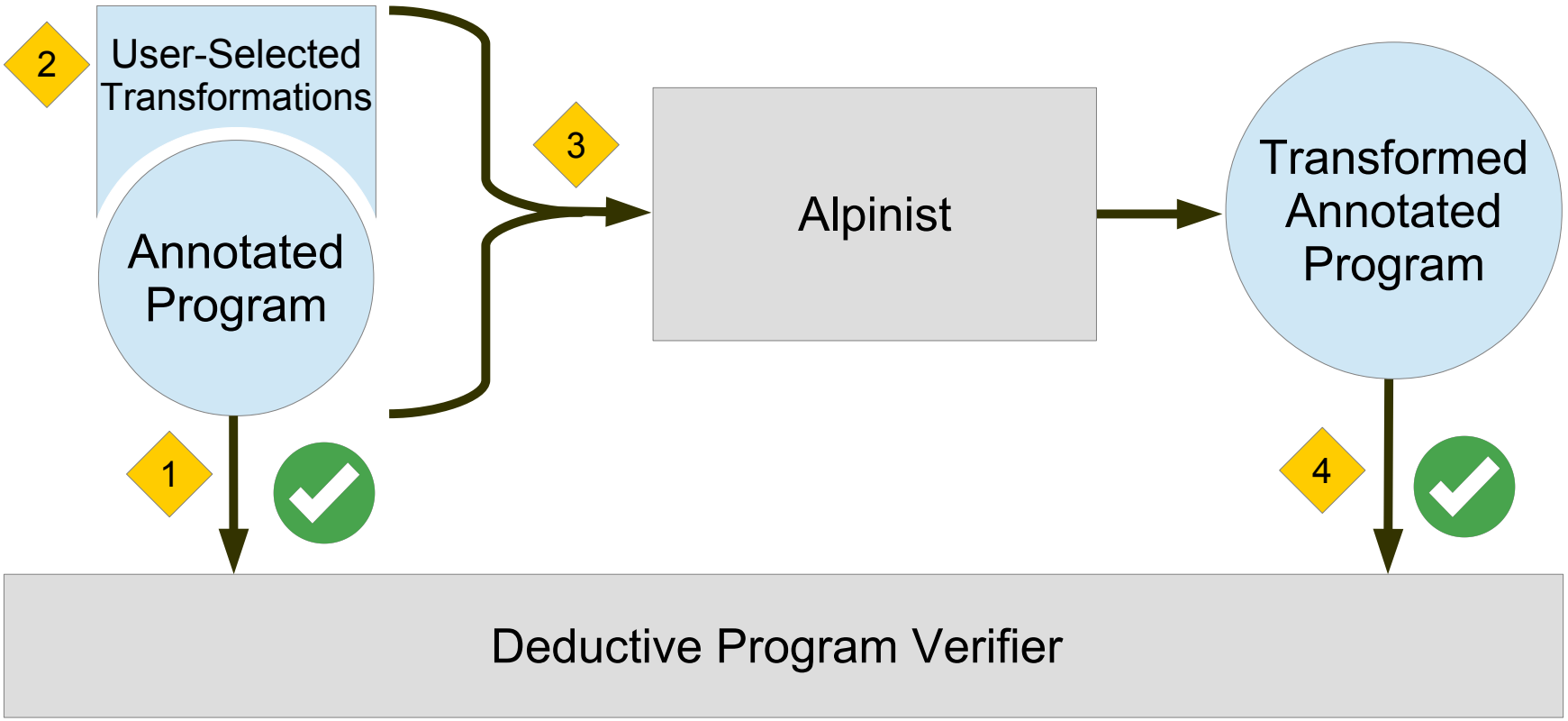
GPU Program Verification



GPU Program Verification



Alpinist an Annotation-Aware GPU Program Optimizer



Abstract GPU Program

```
1  void simple_example(int[] a, int N) {
2    par kernel1 (int tid = 0 .. a.length) {
3      a[tid] = 0;
4    }
5
6    par kernel2(int tid = 0 .. a.length) {
7      for(int k = 0; k < N; k++)
8      {
9        if (tid != a.length-1) {
10         a[tid+1] = a[tid+1] + tid;
11       } else {
12         a[0] = a[0] + tid;
13       }
14     }
15   }
16 }
```

Abstract GPU Program

```
1  void simple_example(int[] a,int N) {
2    par fused_kernels(int fused_tid = 0 .. a.length) {
3      a [ fused_tid ] = 0;
4
5      barrier(fused_kernels)
6
7      for(int k = 0;k < N;k ++){
8        if (fused_tid != a.length - 1) {
9          a [ (fused_tid + 1) ] = a [ (fused_tid + 1) ] + fused_tid;
10       } else {
11         a [ 0 ] = a [ 0 ] + fused_tid;
12       }
13     }
14   }
15 }
```

Abstract GPU Program with Annotations

```
1  ...
2  void simple_example(int[] a, int N) {
3    par kernel1 (int tid = 0 .. a.length)
4      /*@ context Perm(a[tid], 1);
5      ensures a[tid] == 0; @*/
6    { a[tid] = 0; }
7
8    ...
9  } } }
```

Abstract GPU Program with Annotations

```
1  ...
2  void simple_example(int[] a, int N) {
3    par kernel1 (int tid = 0 .. a.length)
4    /*@ context Perm(a[tid], 1);
5    ensures a[tid] == 0; @*/
6    { a[tid] = 0; }
7
8    ...
9  } } }
```

Abstract GPU Program with Annotations

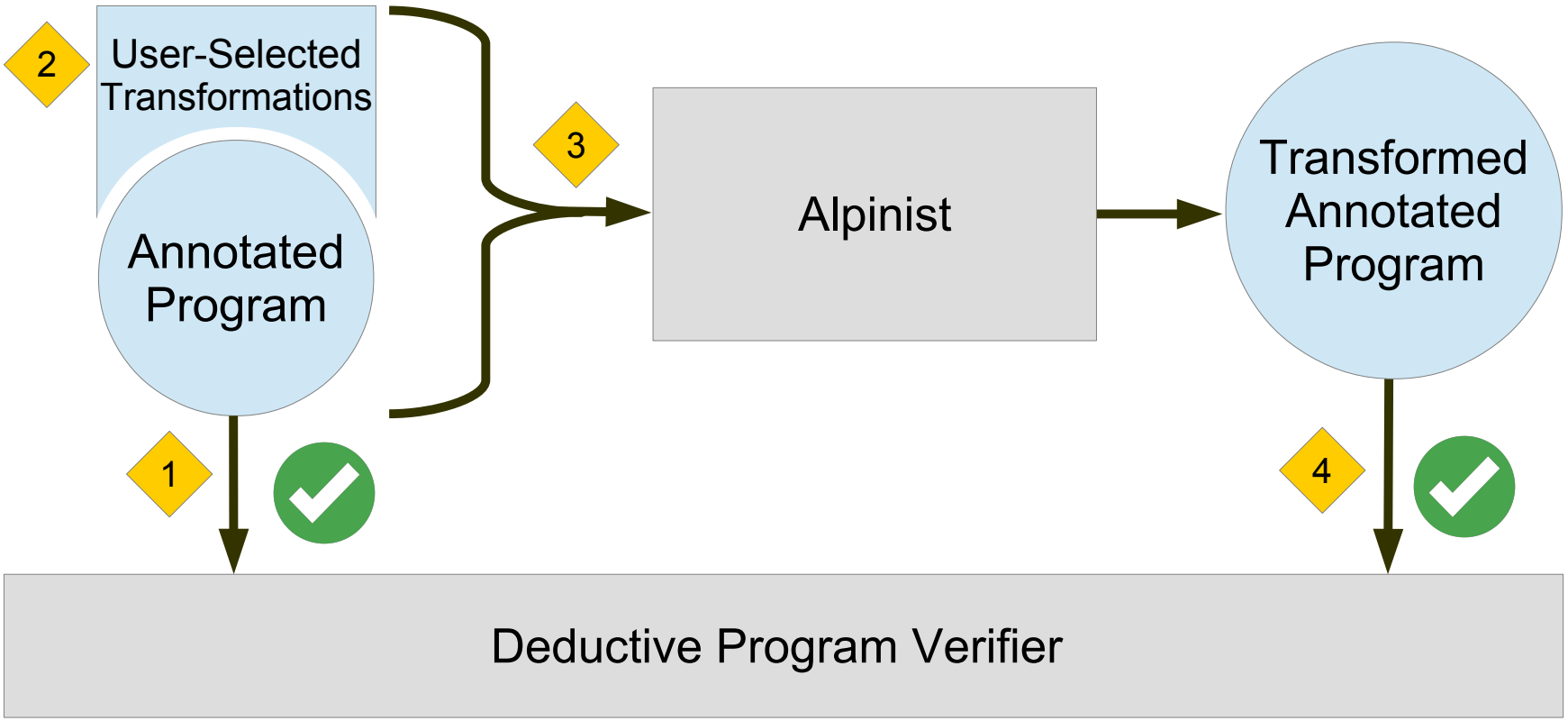
```
1  ...
2  void simple_example(int[] a, int N) {
3    par kernel1 (int tid = 0 .. a.length)
4    /*@ context Perm(a[tid], 1\2);
5    ensures a[tid] == 0; @*/
6    { a[tid] = 0; }
7
8    ...
9  } } }
```

Abstract GPU Program with Annotations

```
1  ...
2  void simple_example(int[] a, int N) {
3    par kernel1 (int tid = 0 .. a.length)
4    /*@ context Perm(a[tid], 1);
5      ensures a[tid] == 0; @*/
6    { a[tid] = 0; }
7
8    ...
9  } } }
```

```
1   ...
2   void simple_example(int[] a, int N) {
3     par kernel2 (int tid = 0 .. a.length)
4     /*@ context Perm(a[tid], 1);
5     ensures a[tid] == 0; @*/
6     { a[tid] = 0; }
7
8     par kernel2 (int tid = 0 .. a.length)
9     /*@ context tid != a.length-1 ? Perm(a[tid+1], 1) : Perm(a[0], 1);
10    requires tid != a.length-1 ? a[tid+1] == 0 : a[0] == 0;
11    ensures tid != a.length-1 ? a[tid+1] == N*tid : a[0] == N*tid; @*/
12    {
13       /*@ loop_inv k >= 0 && k <= N;
14       loop_inv tid != a.length-1 ? Perm(a[tid+1], 1) : Perm(a[0], 1);
15       loop_inv tid != a.length-1 ? a[tid+1] == k*tid : a[0] == k*tid; @*/
16       for(int k = 0; k < N; k++) {
17        if (tid != a.length-1) { a[tid+1] = a[tid+1] + tid; }
18        else { a[0] = a[0] + tid; }
19  } } }
```

ALPINIST an Annotation-Aware GPU Program Optimizer



GPU Optimizations

GPU Optimizations

- Applicability of an optimization
- Complex decisions

Loop unrolling

```
1  void Host(int[] arr, int N){
2    par kernel(tid=0..arr.length){
3      int i = 0;
4      while (i < N){
5        int newInt = i;
6        arr[tid] = arr[tid] + newInt;
7        i = i + 1;
8      }
9    }
10 }
```

```
1  void Host(int[] arr, int N){
2    par kernel(tid=0..arr.length){
3      int i = 0;
4      int newInt = i;
5      arr[tid] = arr[tid] + newInt;
6      i = i + 1;
7
8      newInt = i;
9      arr[tid] = arr[tid] + newInt;
10     i = i + 1;
11
12     while (i < N){
13       newInt = i;
14       arr[tid] = arr[tid] + newInt;
15       i = i + 1;
16     }
17   }
18 }
```

Loop unrolling with annotations

```
1  /*@ context N > 1; @*/
2  void Host(int[] arr, int N){
3    par kernel(tid=0..arr.length){
4      int i = 0;
5      /*@ loop_inv i >= 0 && i <= N;
6        loop_inv N > 1;
7        loop_inv Inv(i); @*/
8      while (i < N){
9        int newInt = i;
10       arr[tid] = arr[tid] + newInt;
11       i = i + 1;
12     }
13   }
14 }
```

```
1  /*@ context N > 1; @*/
2  void Host(int[] arr, int N){
3    par kernel(tid=0..a.length){
4      int i = 0;
5      int newInt = i;
6      arr[tid] = arr[tid] + newInt;
7      i = i + 1;
8      //@ assert i >= 1 && i <= N;
9      //@ assert N > 1;
10     //@ assert Inv(i);
11     newInt = i;
12     arr[tid] = arr[tid] + newInt;
13     i = i + 1;
14     /*@ loop_inv i >= 2 && i <= N;
15      loop_inv N > 1;
16      loop_inv Inv(i); @*/
17     while (i < N){
18       newInt = i;
19       arr[tid] = arr[tid] + newInt;
20       i = i + 1;
21     }
22   }
23 }
```

Kernel fusion

```
1  void Host(...){
2    par kernel1(tid1 = 0..T) {
3      a[tid1] = 2*b[tid1];
4    }
5
6    par kernel2(tid2 = 0..T) {
7      b[tid2] = a[tid2]+1;
8    }
9  }
```

```
1  void Host(...){
2    par fused_kernels(tid = 0..T) {
3      a[tid] = 2*b[tid];
4      b[tid] = a[tid]+1;
5    }
6  }
```

Kernel fusion with annotations

```
1   void Host(...){
2     par kernel1(tid1 = 0..T)
3     /*@ context Perm(a[tid1], 1);
4     context Perm(b[tid1], 1\2);@*/
5     {
6       a[tid1] = 2*b[tid1];
7     }
8
9     par kernel2(tid2 = 0..T)
10    /*@ context Perm(a[tid2], 1\2);
11    context Perm(b[tid2], 1);@*/
12    {
13      b[tid2] = a[tid2]+1;
14    }
15  }
```

Kernel fusion with annotations

```
1  void Host(...){
2    par kernel1(tid1 = 0..T)
3    /*@ context Perm(a[tid1], 1);
4    context Perm(b[tid1], 1\2);@*/
5    {
6      a[tid1] = 2*b[tid1];
7    }
8
9    par kernel2(tid2 = 0..T)
10   /*@ context Perm(a[tid2], 1\2);
11   context Perm(b[tid2], 1);@*/
12   {
13     b[tid2] = a[tid2]+1;
14   }
15 }
```

```
1  void Host(...){
2    par fused_kernels(tid = 0..T)
3    /*@ requires Perm(a[tid], 1);
4    requires Perm(b[tid], 1\2);
5    requires Perm(b[tid], 1\2);@*/
6    {
7      a[tid] = 2*b[tid];
8      b[tid] = a[tid]+1;
9    }
10 }
```

```
1  ...
2  void simple_example(int[] a, int size, int N) {
3    par kernel1 (int tid = 0 .. a.length)
4    /*@ context Perm(a[tid], 1);
5    ensures a[tid] == 0; @*/
6    { a[tid] = 0; }
7
8    par kernel2 (int tid = 0 .. a.length)
9    /*@ context tid != a.length-1 ? Perm(a[tid+1], 1) : Perm(a[0], 1);
10   requires tid != a.length-1 ? a[tid+1] == 0 : a[0] == 0;
11   ensures tid != a.length-1 ? a[tid+1] == N*tid : a[0] == N*tid; @*/
12   {
13     /*@ loop_inv k >= 0 && k <= N;
14     loop_inv tid != a.length-1 ? Perm(a[tid+1], 1) : Perm(a[0], 1);
15     loop_inv tid != a.length-1 ? a[tid+1] == k*tid : a[0] == k*tid; @*/
16     for(int k = 0; k < N; k++) {
17       if (tid != a.length-1) { a[tid+1] = a[tid+1] + tid; }
18       else { a[0] = a[0] + tid; }
19 } } }
```

```
1  ...
2  void simple_example(int[] a, int size, int N) {
3    par kernel1 (int tid = 0 .. a.length)
4    /*@ context Perm(a[tid], 1) ;
5    ensures a[tid] == 0; @*/
6    { a[tid] = 0; }
7
8    par kernel2 (int tid = 0 .. a.length)
9    /*@ context tid != a.length-1 ? Perm(a[tid+1], 1) : Perm(a[0], 1);
10   requires tid != a.length-1 ? a[tid+1] == 0 : a[0] == 0;
11   ensures tid != a.length-1 ? a[tid+1] == N*tid : a[0] == N*tid; @*/
12   {
13     /*@ loop_inv k >= 0 && k <= N;
14     loop_inv tid != a.length-1 ? Perm(a[tid+1], 1) : Perm(a[0], 1);
15     loop_inv tid != a.length-1 ? a[tid+1] == k*tid : a[0] == k*tid; @*/
16     for(int k = 0; k < N; k++) {
17       if (tid != a.length-1) { a[tid+1] = a[tid+1] + tid; }
18       else { a[0] = a[0] + tid; }
19 } } }
```

Decision points in the perm-related precondition procedure

1. Patterns are the same (i.e. thread to location mapping the same)

Decision points in the perm-related precondition procedure

1. Patterns are the same (i.e. thread to location mapping the same)
2. Permission for a location ≤ 1

Decision points in the perm-related precondition procedure

1. Patterns are the same (i.e. thread to location mapping the same)
2. Permission for a location ≤ 1
3. Only read permissions

Decision points in the perm-related precondition procedure

1. Patterns are the same (i.e. thread to location mapping the same)
2. Permission for a location ≤ 1
3. Only read permissions
4. One of the kernels has write permissions

Decision points in the perm-related precondition procedure

1. Patterns are the same (i.e. thread to location mapping the same)
2. Permission for a location ≤ 1
3. Only read permissions
4. One of the kernels has write permissions

Decision points in the perm-related precondition procedure

1. Patterns are the same (i.e. thread to location mapping the same)
2. Permission for a location ≤ 1
3. Only read permissions
4. One of the kernels has write permissions, data dependency

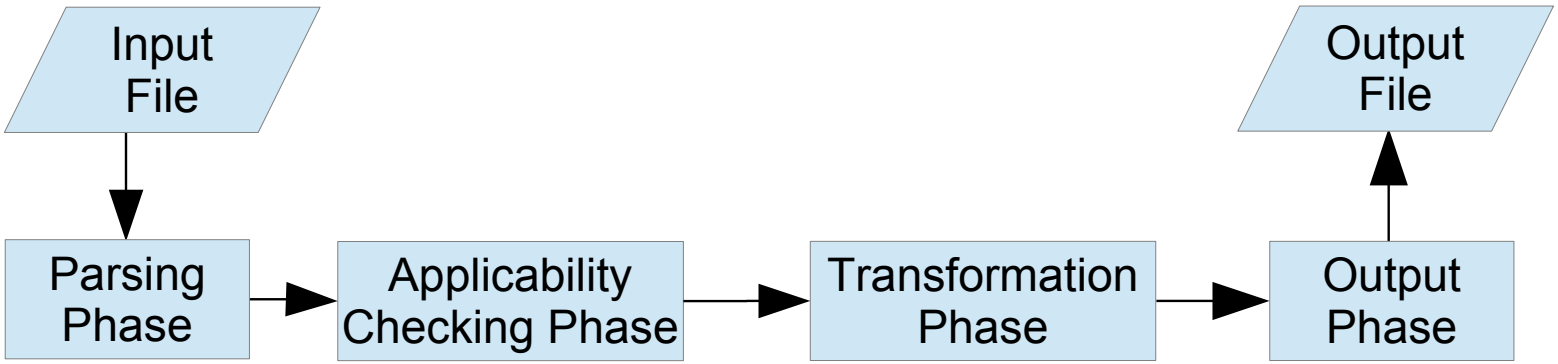
Supported GPU Optimizations in Alpinist

- Loop unrolling
- Kernel fusion
- Tiling
- Iteration merging
- Matrix linearization
- Data prefetching

Supported GPU Optimizations in Alpinist

- Loop unrolling
- Kernel fusion
- Tiling
- Iteration merging
- Matrix linearization
- Data prefetching

(Internal) Design of the Tool



Evaluation

Evaluation

- **Q1** test whether Alpinist works on GPU programs.

Evaluation

- **Q1** test whether ALPINIST works on GPU programs.
- **Q2** investigate how long it takes for ALPINIST to transform GPU programs and how this affects the verification time.

Evaluation

- **Q1** test whether Alpinist works on GPU programs.
- **Q2** investigate how long it takes for Alpinist to transform GPU programs and how this affects the verification time.
- **Q3** investigate the usability of Alpinist on real-world complex examples.

Q1 & Q3 Evaluation Suite

Q1 & Q3 Evaluation Suite

1. Hand-made examples

Q1 & Q3 Evaluation Suite

1. Hand-made examples
2. Verified examples in VerCors

Q1 & Q3 Evaluation Suite

1. Hand-made examples
2. Verified examples in VerCors
3. Complex case studies in VerCors:
   - Two parallel prefix sum algorithms
   - Parallel stream compaction
   - Summed-area table algorithms
   - A variety of sorting algorithms
   - A solution to VerifyThis 2019 challenge 1
   - A Tic-Tac-Toe example

Q1 & Q3 Evaluation Suite

1. Hand-made examples
2. Verified examples in VerCors
3. Complex case studies in VerCors:
   - Two parallel prefix sum algorithms
   - Parallel stream compaction
   - Summed-area table algorithms
   - A variety of sorting algorithms
   - A solution to VerifyThis 2019 challenge 1
   - A Tic-Tac-Toe example

70 experiments in total across all 6 optimizations

Q2 Effect on verification time

Optimizations	Effect on Code/Annotations	Avg. Verification Time

Q2 Effect on verification time

Optimizations	Effect on Code/Annotations	Avg. Verification Time
Loop unrolling	**Generates** code/annotations	+14%
Tiling	**Generates** code/annotations	+32%
Iteration merging	**Generates** code/annotations	+18%

Q2 Effect on verification time

Optimizations	**Effect on Code/Annotations**	**Avg. Verification Time**
Loop unrolling	**Generates** code/annotations	+14%
Tiling	**Generates** code/annotations	+32%
Iteration merging	**Generates** code/annotations	+18%
Kernel Fusion	**Removes** code/annotations	-23%

Q2 Effect on verification time

Optimizations	Effect on Code/Annotations	Avg. Verification Time
Loop unrolling	**Generates** code/annotations	+14%
Tiling	**Generates** code/annotations	+32%
Iteration merging	**Generates** code/annotations	+18%
Kernel Fusion	**Removes** code/annotations	-23%
Matrix Linearization	**Replaces** code/annotations	+1%
Data prefetching	**Replaces** code/annotations	-4%

To conclude

- ALPINIST: an Annotation-Aware GPU Program Optimizer

To conclude

- ALPINIST: an Annotation-Aware GPU Program Optimizer
- Six optimizations supported in the tool

To conclude

- ALPINIST: an Annotation-Aware GPU Program Optimizer
- Six optimizations supported in the tool
- Future work

To conclude

- ALPINIST: an Annotation-Aware GPU Program Optimizer
- Six optimizations supported in the tool
- Future work
  - More GPU optimizations

To conclude

- Alpinist: an Annotation-Aware GPU Program Optimizer
- Six optimizations supported in the tool
- Future work
  - More GPU optimizations
  - Support for OpenCL and CUDA

Alpinist an Annotation-Aware GPU Program Optimizer