Improved Static Analysis to Generate More Efficient Code for Execution of Loop Nests in GPUs

J. Nelson Amaral

Department of Computing Science

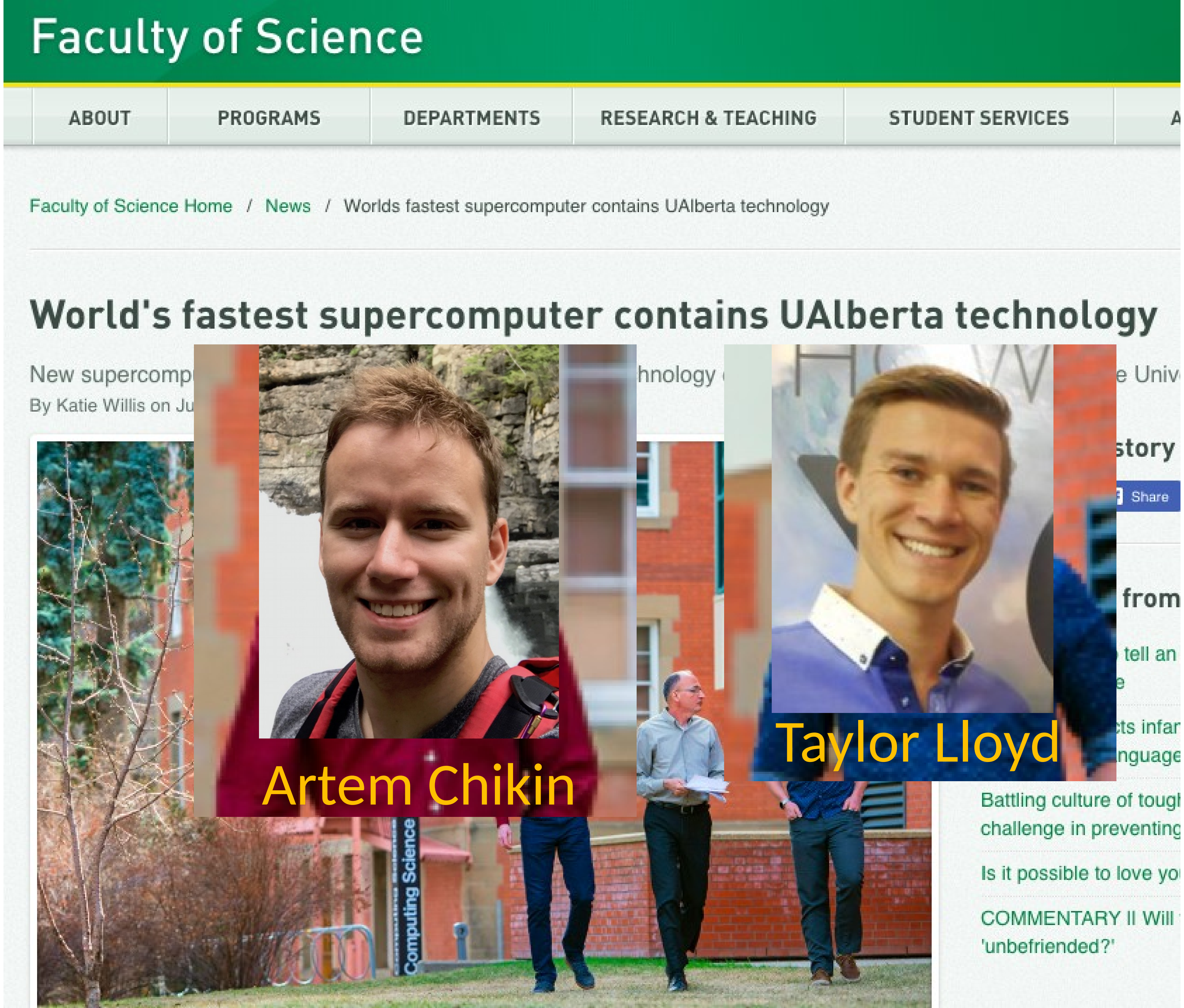
UNIVERSITY OF ALBERTA


Michael N.
Nathan
Michael Z
Jacky
Braedy
Sarah
Rebecca
Eldon
Dylan
Thomas
Ben Hao
Ben Hao
Tristan
Brad Simons
Artem Chikin
Lyle Roleman
Artem H.

September 2014


Antem Chikin

September 2014 July 2018

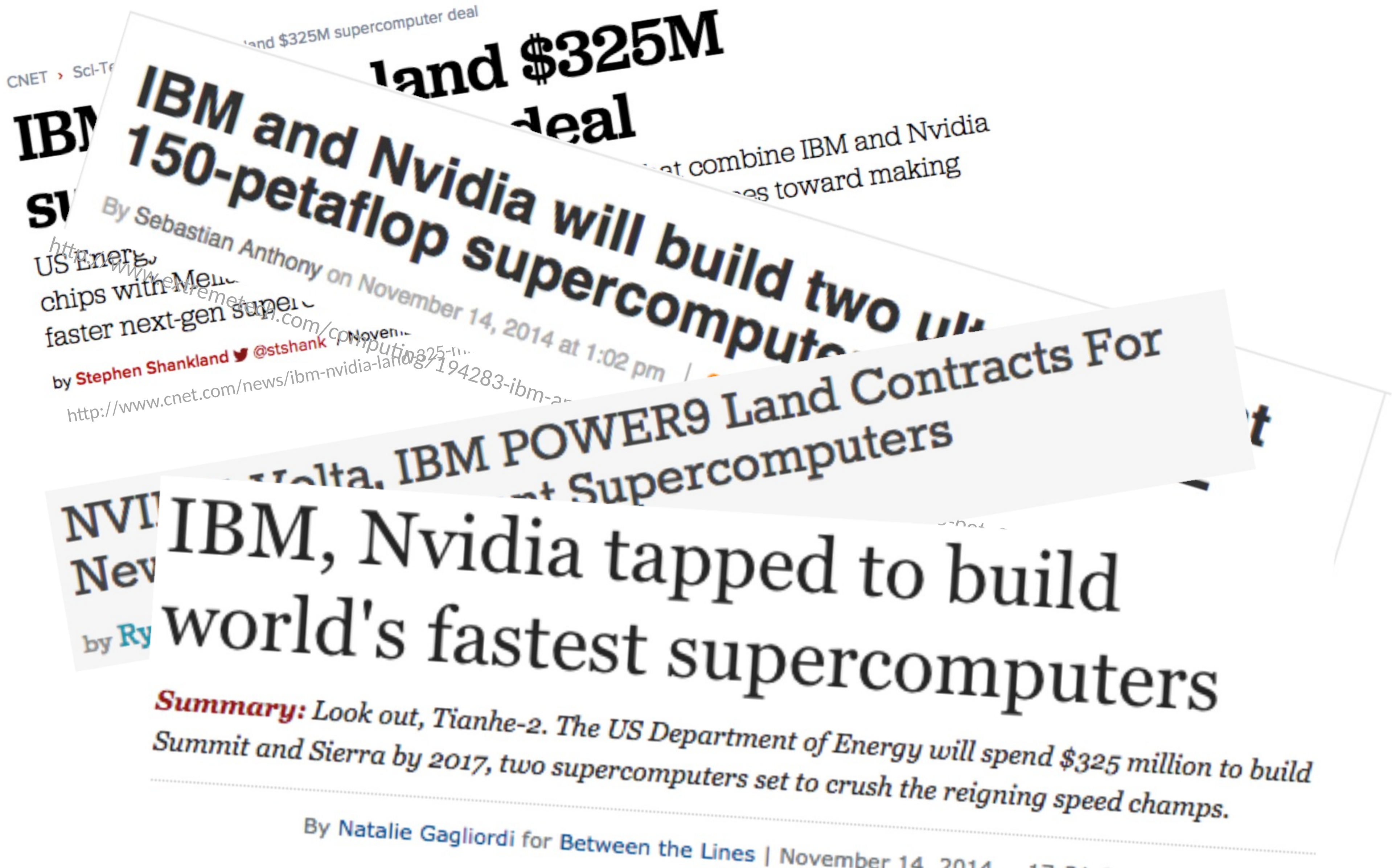

Faculty of Science
ABOUT
PROGRAMS
DEPARTMENTS
RESEARCH & TEACHING
STUDENT SERVICES
Faculty of Science Home / News / Worlds fastest supercomputer contains UAlberta technology
World's fastest supercomputer contains UAlberta technology
Artem Chikin
Taylor Lloyd
Share
Battling culture of tough
challenge in preventing
Is it possible to love yo
COMMENTARY II Will
'unbefriended?'
Computing Science

IBM and the DoE launch the world's fastest supercomputer

Frederic Lardinois @fredericl / Jun 8, 2018

Comment

November 2014



http://www.zdnet.com/article/ibm-nvidia-tapped-to-build-worlds-fastest-supercomputers/

RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)	RPEAK (TFLOP/S)	POWER (KW)
1	National Super Computer	Tianhe-2 (MilkyWay-2) -		33,862.7	54,902.4	17,808
2			560,640	17,590.0	27,112.5	8,209
3	United States	Custom	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advan Computation Japan		705,024	10,5		
5	DOE/SC/Arg Laboratory United State		786,432	8,586.6		
6	Swiss Nation Supercom (CSCS) Switzerland		115,98			
7	Texas Advan Center/Univ. United States	Dell	462,462	5,168.1	8,520.1	4,510
8	Forschungszentrum Juelich (FZJ) Germany	JUQUEEN - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
9	DOE/NNSA/LLNL United States	Vulcan - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
10	Government United States	Cray CS-Storm, Intel Xeon E5-2660v2 10C 2.2GHz, Infiniband FDR, Nvidia K40 Cray Inc.	72,800	3,577.0	6,131.8	1,499

200000 Linpack TFLOP/S

~~88000 Linpack TFLOP/S~~

17,590.0

8,209

15000 KW

~~9000 KW~~

"Summit … is expected to deliver more than five times the system-level application performance of Titan while consuming only 10% more power."

http://info.nvidianews.com/rs/nvidia/images/Coral%20White%20Paper%20Final-3-2.pdf

http://www.top500.org/list/2014/11/

Technology?

Nvidia Volta GPU

IBM Power9

Nvidia NVlink

Programming Model?

OpenMP

~~OpenACC~~

~~MPI~~

~~OpenCL~~

~~CUDA~~

Compiler Technology?

LLVM

IBM XL Compiler

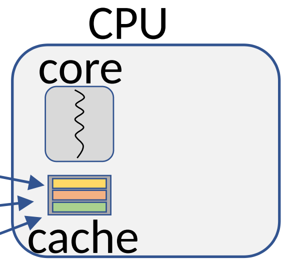
May 2015


Taylor Lloyd
J. Nelson Amaral
Ettore Tiotto
Artem Chikin
UNIVERSITY OF ALBERTA
Science Internship Program
IBM Canada Software Laboratory
Markham, ON

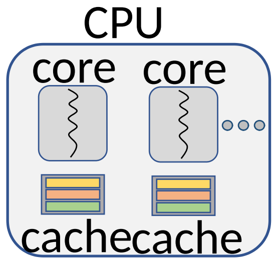
Programming Model

OpenMP 3.x ➔ OpenMP 4.x

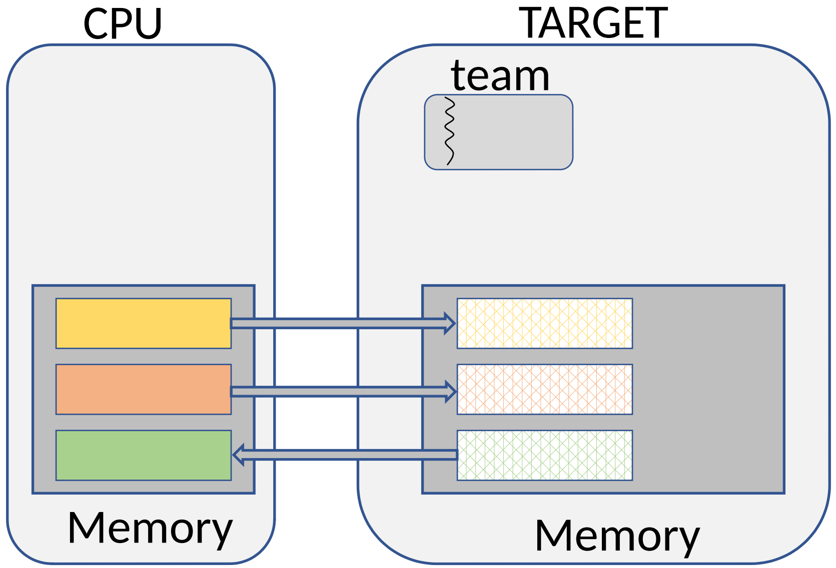
```
void vecAdd(double *a, double *b, double *c, int n)
{
   for (int i = 0; i < n; i++) {
      c[i] = a[i] + b[i];
   }
}
```



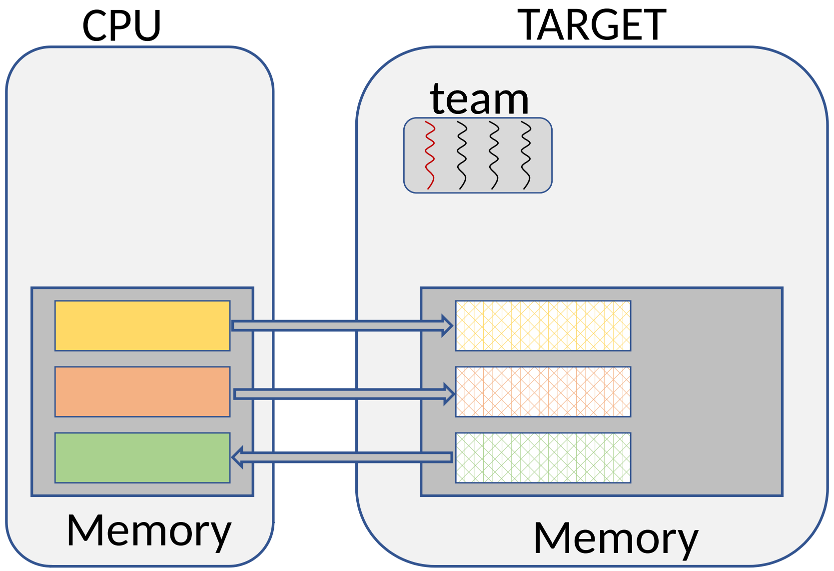

```
void vecAdd(double *a, double *b, double *c, int n)
{
   #pragma omp parallel for
   for (int i = 0; i < n; i++) {
      c[i] = a[i] + b[i];
   }
}
```



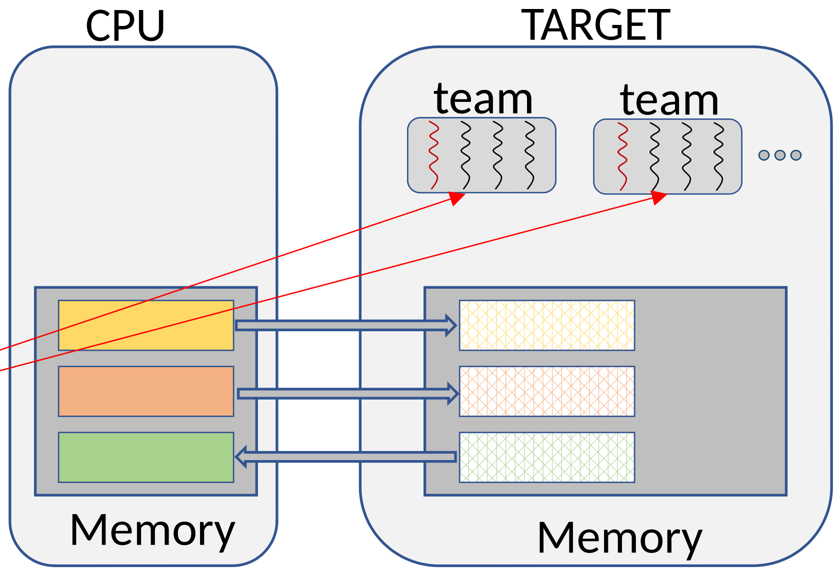

```
void vecAdd(double *a, double *b, double *c, int n)
{
   #pragma omp target map(to: a[:n], b[:n])
                    \ map(from: c[:n])
   for (int i = 0; i < n; i++) {
      c[i] = a[i] + b[i];
   }
}
```



```c
void vecAdd(double *a, double *b, double *c, int n)
{
  #pragma omp target map(to: a[:n], b[:n])
                \ map(from: c[:n])
  #pragma omp parallel for
  for (int i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
  }
}
```

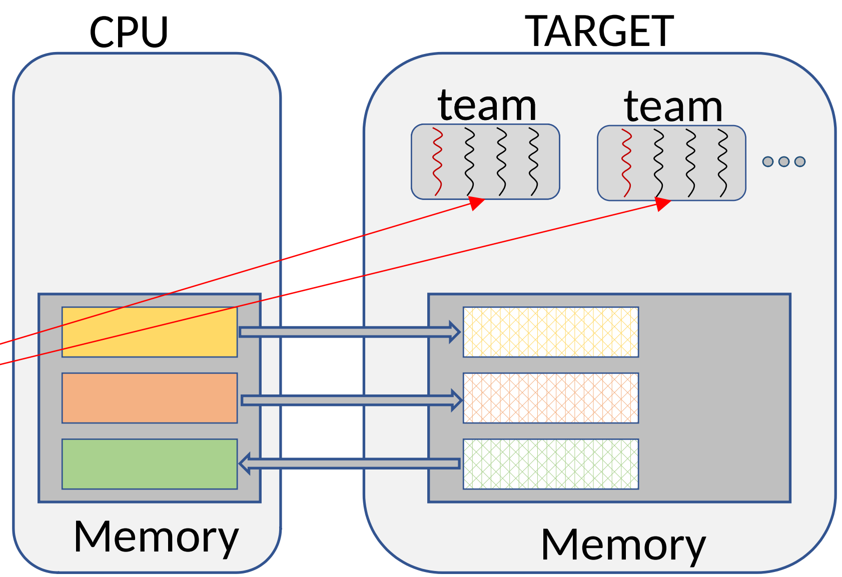



```c
void vecAdd(double *a, double *b, double *c, int n)
{
  #pragma omp target map(to: a[:n], b[:n])
                \ map(from: c[:n])
  #pragma omp teams parallel for
  for (int i = 0; i < n; i++) {
    c[i] = a[i] + b[i];
  }
}
```



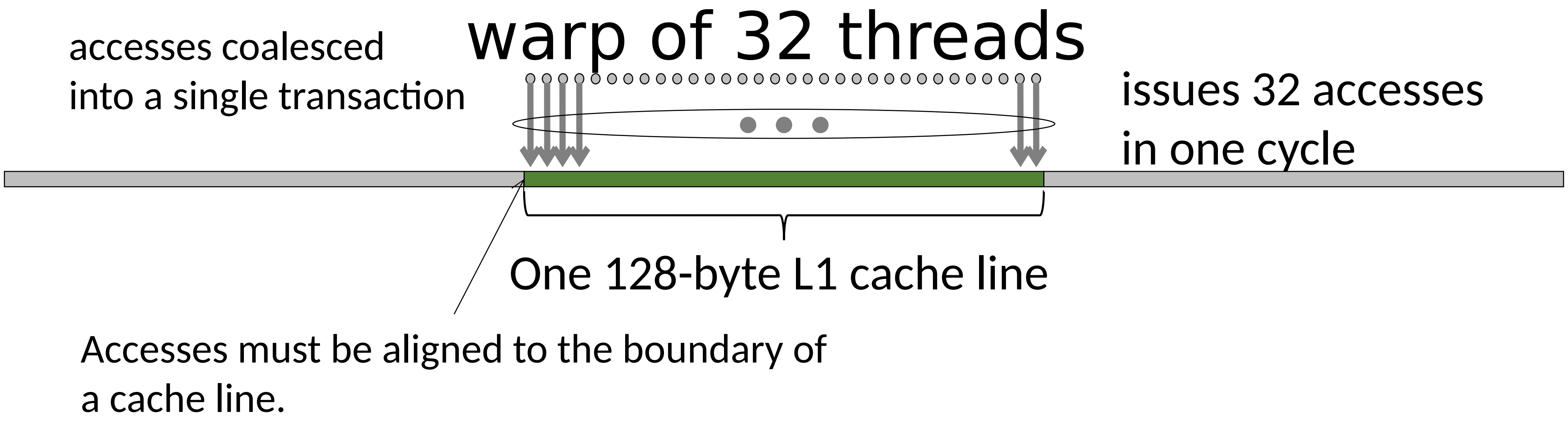
This two teams are executing the same computation on the same memory locations without synchronization. Likely wrong result.

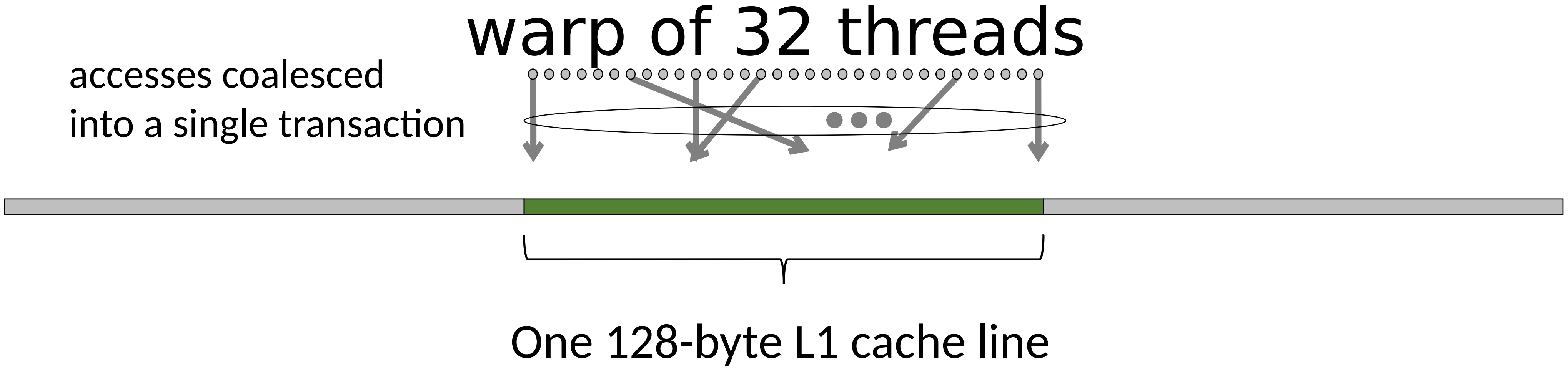
```
void vecAdd(double *a, double *b, double *c, int n)
{
   #pragma omp target map(to: a[:n], b[:n])
                  \ map(from: c[:n])
   #pragma omp teams distribute parallel for
   for (int i = 0; i < n; i++) {
     c[i] = a[i] + b[i];
   }
}
```



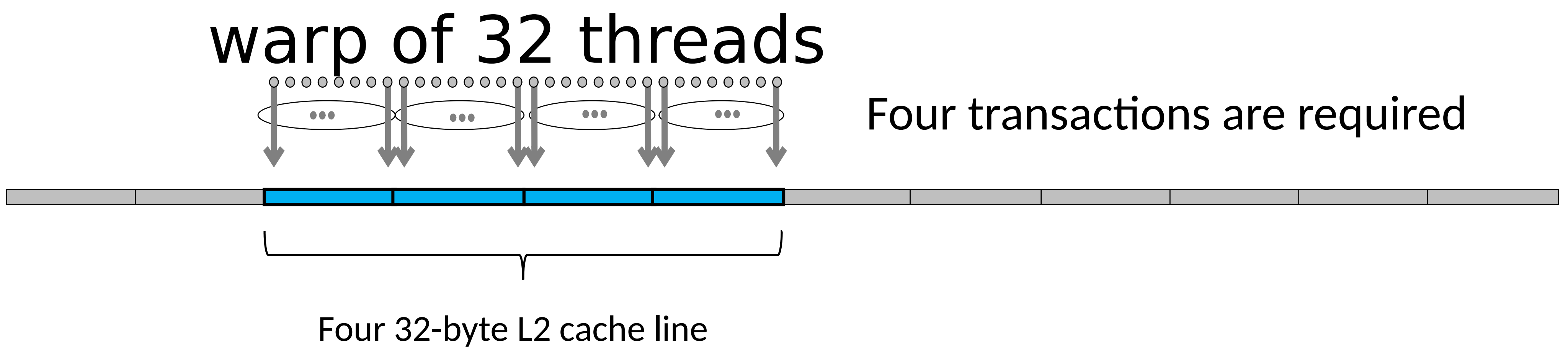
The iterations of the loop are distributed to the two teams and the result is correct.

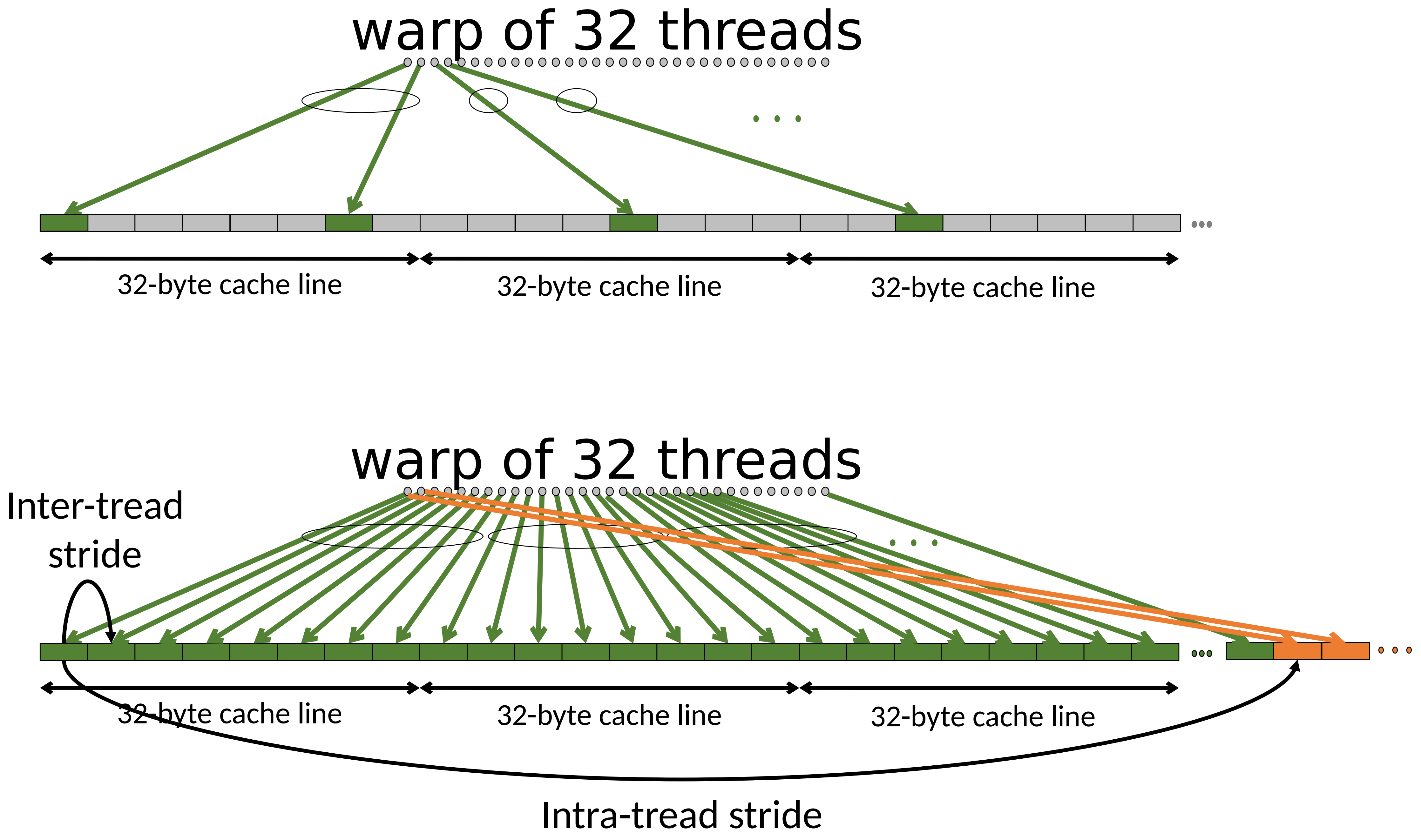
Memory Coalescing





Warps may access addresses in any order within the cache line.

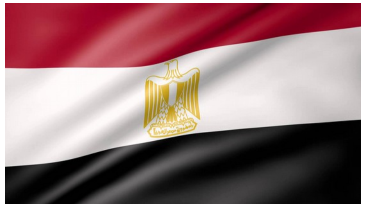

warp of 32 threads
Four transactions are required
Four 32-byte L2 cache line


warp of 32 threads
32-byte cache line
32-byte cache line
32-byte cache line
warp of 32 threads
Inter-tread
stride
32-byte cache line
32-byte cache line
32-byte cache line
Intra-tread stride

October 2016

Taylor Lloyd

Karim Ali



Taint Analysis

a is tainted

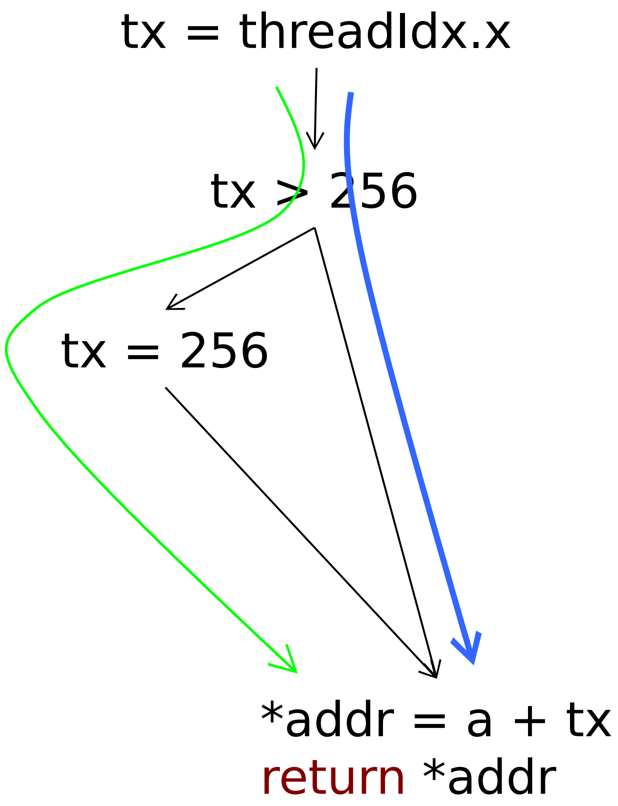
```
void main() {
      long int a = readCreditCardNumber();
      long int b = 0;

      b = foo(a);

      print(b);
}
```

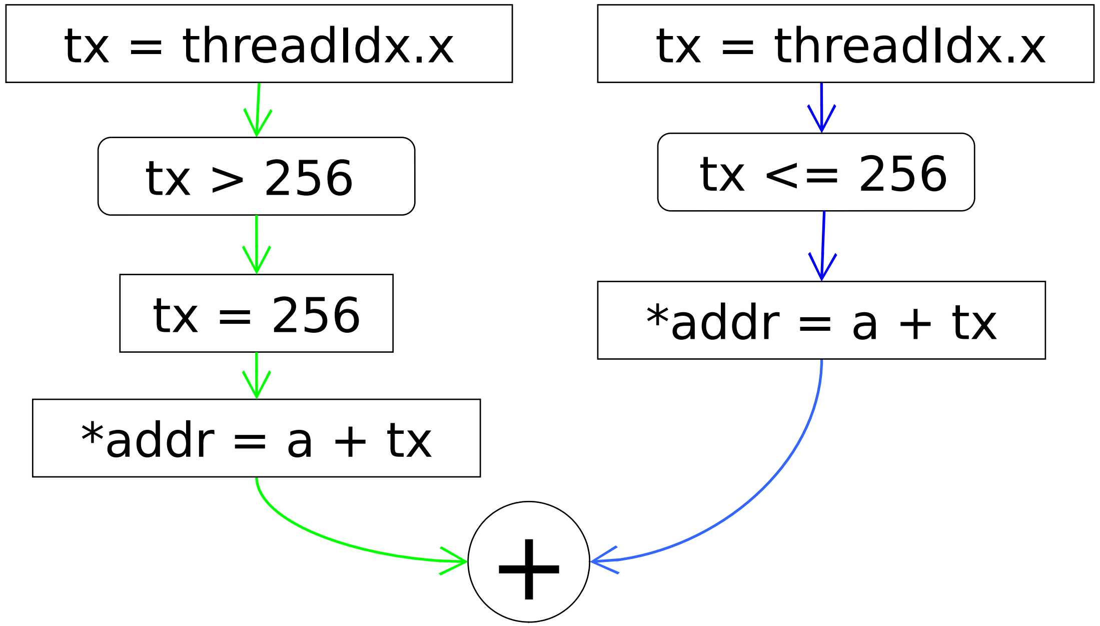
is b tainted?

```
long int foo(int p) {

      if(p != 0)
        print(p);

}
```

Taylor Lloyd

Arithmetic Control Form (ACF) Analysis






```
int readBounded(int* a)
{
    int tx = threadIdx.x;
    if(tx > 256)
        tx = 256;
    int *addr = a + tx;
    return *addr;
}
```

$$ACF_T(*addr) = (T > 256) \; *([a] + 4*256) \; + \; (T <= 256) \; *([a] + 4*T)$$

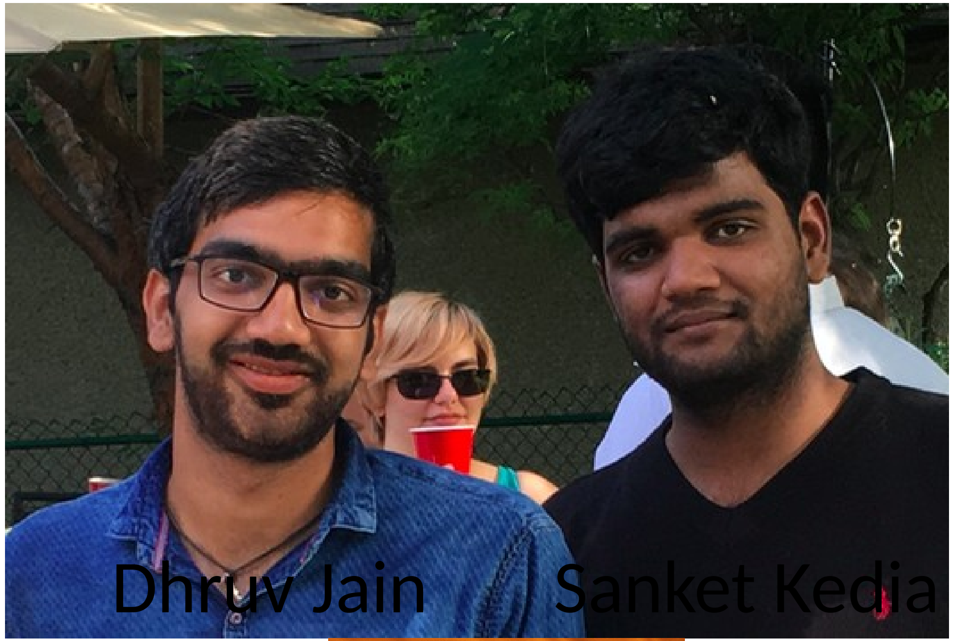
$$ACF_0(*addr) = (0 > 256)*([a] + 4*256) + (0 <= 256)*([a] + 4*0)$$

$$ACF_0(*addr) = [a]$$

$$ACF_1(*addr) = [a] + 4$$

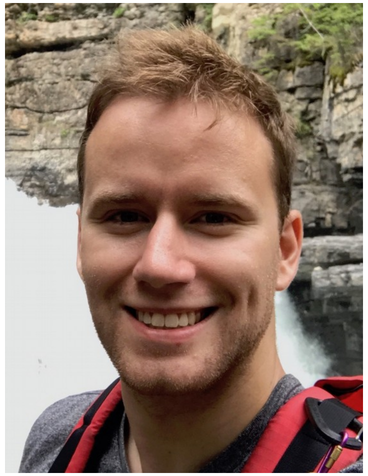
$$ACF_1(*addr) – ACF_0(*addr) = 4$$

June/July 2017

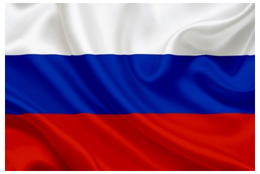


Dhruv Jain Sanket Kedia

Taylor Lloyd



Artem Chikin



Automated GPU Grid Geometry Selection for OpenMP Kernels

Taylor Lloyd
University of Alberta
Edmonton, Canada
tjlloyd@ualberta.ca

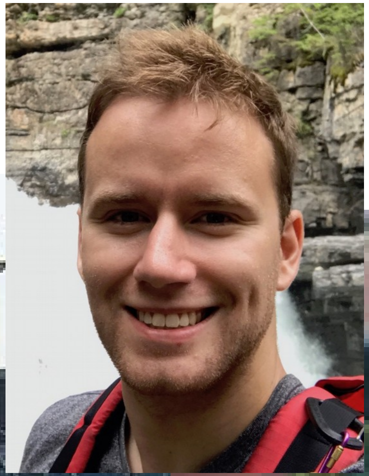
Artem Chikin
University of Alberta
Edmonton, Canada
artem@ualberta.ca

Sanket Kedia
IIT Kharagpur
Kharagpur, India
kedia.sanket@cse.iitkgp.ernet.in

Dhruv Jain
IIT Kharagpur
Kharagpur, India
dhruvjaincse@iitkgp.ac.in

José Nelson Amaral
University of Alberta
Edmonton, Canada
jamaral@ualberta.ca

August 2017



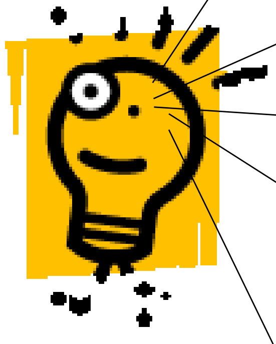
Artem Chikin

IBM Canada Software Laboratory

Markham, ON

Computing Science Centre

Iteration Point Difference Analysis (IPDA)



- ACF can be used for any pair of expressions.
- ACF can be based on the induction variables in a loop nest.
- ACF is useful when applied to address expressions in a loop nest.
- ACF can make a Data Dependence Graph more precise.
- Compiler can transform code based on IPDA.

IPDA Analysis in an example

PolyBench/GPU

Implementation of PolyBench codes for GPU processing

[<< home] [news] [description] [download] [documentation] *Version 1.0 available*

News

- **03/19/12: Public release of PolyBench/GPU 1.0 Download**

Description

PolyBench/GPU is a collection of PolyBench codes (as well as convolution) implemented for processing on the GPU using CUDA, OpenCL, and HMPP (pragma-based compiler).

- Codes are run on both CPU and GPU for run-time and output comparison
- Info on HMPP available at http://www.caps-entreprise.com/hmpp.html

conv2D: Two-dimensional convolution

```
for (CIVI = 0; CIVI < NI - 2; ++CIVI)
 {
  i = CIVI+1;
  for (CIVJ = 0; CIVJ < NJ - 2; ++CIVJ)
  {
   B[i*NJ + CIVJ + 1] = ... ;
  }
 }
```

```
B + 8*((CIVI+1)*NJ + CIVJ + 1)
```

Base address for array B

IPAD propagates symbolic expressions from dominant definition to each use.

Assuming that data type size is 8 bytes

```
for (CIVI = 0; CIVI < NI - 2; ++CIVI)
 {
  i = CIVI+1;
  for (CIVJ = 0; CIVJ < NJ - 2; ++CIVJ)
  {
   B[i*NJ + CIVJ + 1] = ... ;
  }
 }
```

B + 8*((CIVI+1)*NJ + CIVJ + 1)

Iteration Point Algebraic Difference:

B + 8*((CIVI’+1)*NJ + CIVJ’ + 1) - (B + 8*((CIVI+1)*NJ + CIVJ + 1))

8*(CIVI’*NJ + CIVJ’) - 8*(CIVI*NJ + CIVJ)

8*((CIVI’-CIVI)*NJ + (CIVJ’-CIVJ))

8*(ΔCIVI*NJ + ΔCIVJ) = 0 ?

```
for (CIVI = 0; CIVI < NI - 2; ++CIVI)
 {
  i = CIVI+1;
  for (CIVJ = 0; CIVJ < NJ - 2; ++CIVJ)
  {
   B[i*NJ + CIVJ + 1] = ... ;
  }
 }
```

B + 8*((CIVI+1)*NJ + CIVJ + 1)

Iteration Point Algebraic Difference:

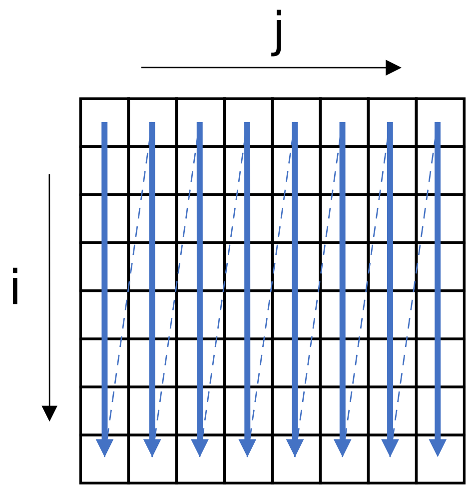
ΔCIVI	ΔCIVJ
$= 0$	$\neq 0$
$\neq 0$	$= 0$
$\neq 0$	$\neq 0$

$8*(\Delta CIVI*NJ + \Delta CIVJ) \quad = 0$?

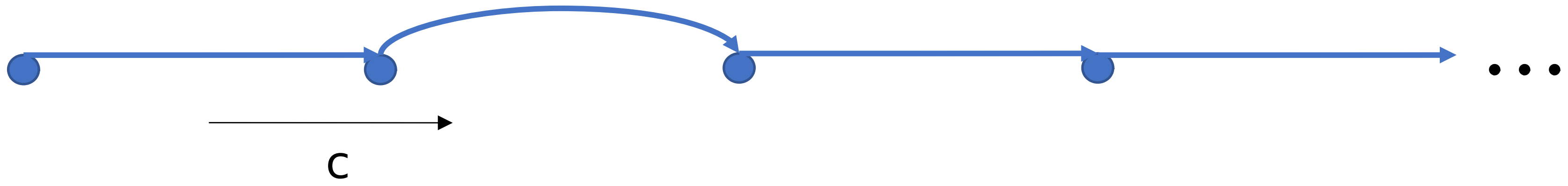
Loop Collapsing and Loop Interchange

```
for (i = 0; i < N; ++i)
 {
   for (j = 0; j < N; ++j)
   {
     A[i+j*N] = A[i+j*N] * A[i+j*N];
   }
 }
```

Loop Collapse




```
for (c = 0; c < N*N; ++c)
 {
   i = c / N;
   j = c % N;
   A[i+j*N] = A[i+j*N] * A[i+j*N];
 }
```



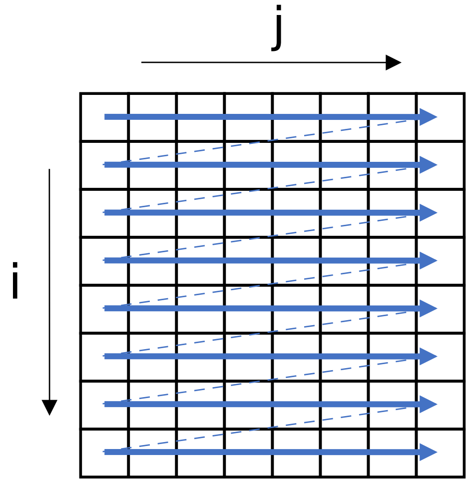
```
for (i = 0; i < N; ++i)
 {
   for (j = 0; j < N; ++j)
   {
     A[i+j*N] = A[i+j*N] * A[i+j*N];
   }
 }
```

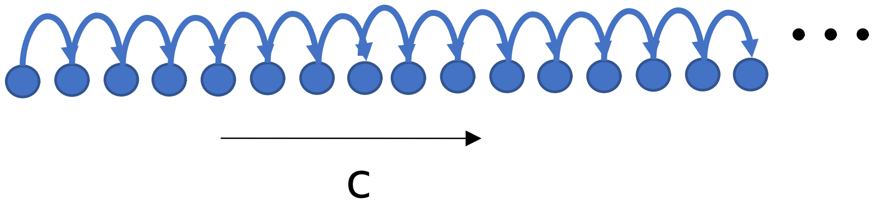
Loop Interchange

```
for (j = 0; j < N; ++j)
 {
   for (i = 0; i < N; ++i)
   {
     A[i+j*N] = A[i+j*N] * A[i+j*N];
   }
 }
```

Loop Collapse

```
for (c = 0; c < N*N; ++c)
 {
   i = c / N;
   j = c % N;
   A[i+j*N] = A[i+j*N] * A[i+j*N];
 }
```





A detailed example of how IPDA Analysis helps

Standard Performance Evaluation Corporation

Results

- Published Results
- Fair Use Policy

Information

- SPEC ACCEL
- Documentation
  - Documentation Overview
  - FAQ
  - Run & Reporting Rules
  - Technical Support
- HPG

Press and Publications

- SPEC ACCEL Release
- SPEC ACCEL V1.2 Release

Order Benchmarks

- Order SPEC ACCEL

Resources

SPEC ACCEL™

557.pcsp

The SPEC ACCEL™ benchmark suite tests perf applications running under the OpenCL, OpenA suite exercises the performance of the accelerator, host CPU, memory transfer between host and accelerator, support libraries and drivers, and compilers.

The SPEC ACCEL™ benchmark is available for purchase via the SPEC order form.

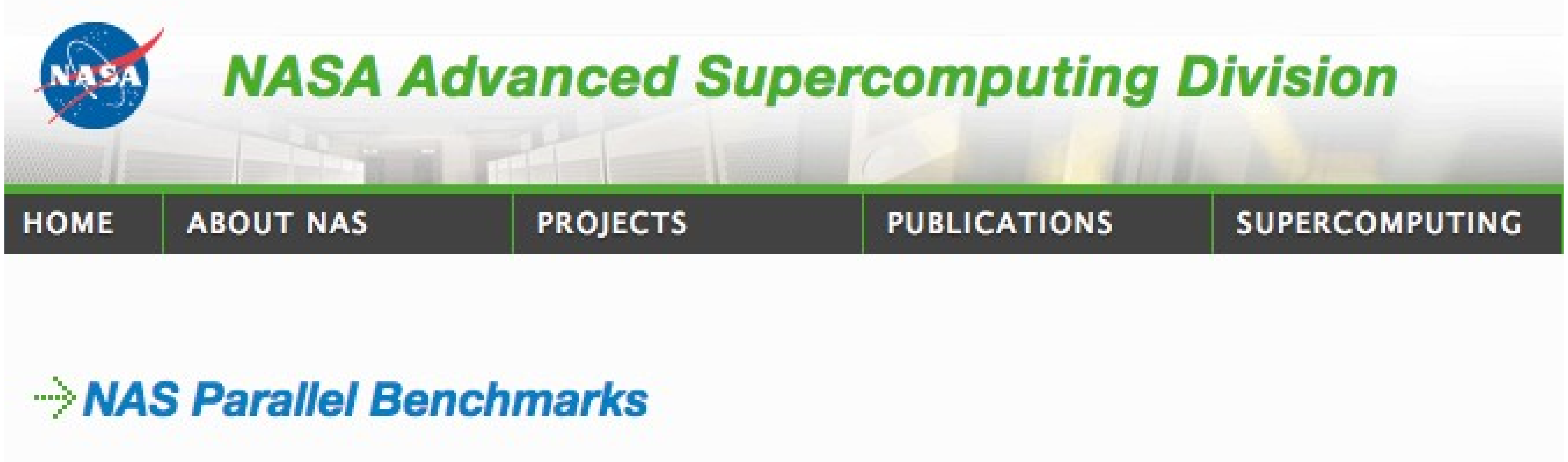
June 21, 2017: SPEC HPG announces an update to the SPEC ACCEL™ benchmark. The **V1.2 update** includes a new suite using OpenMP 4 target directives, bug fixes, improved documentation and PTDaemon 1.8.1 and is available as of June 21, 2017. Results from V1.1 will be accepted until the review cycle starting August 2, 2017. After that date, all submissions must be made using V1.2. **Please note that results for the OpenACC suite from version 1.2 cannot be compared to results using version 1.1 or 1.0.**

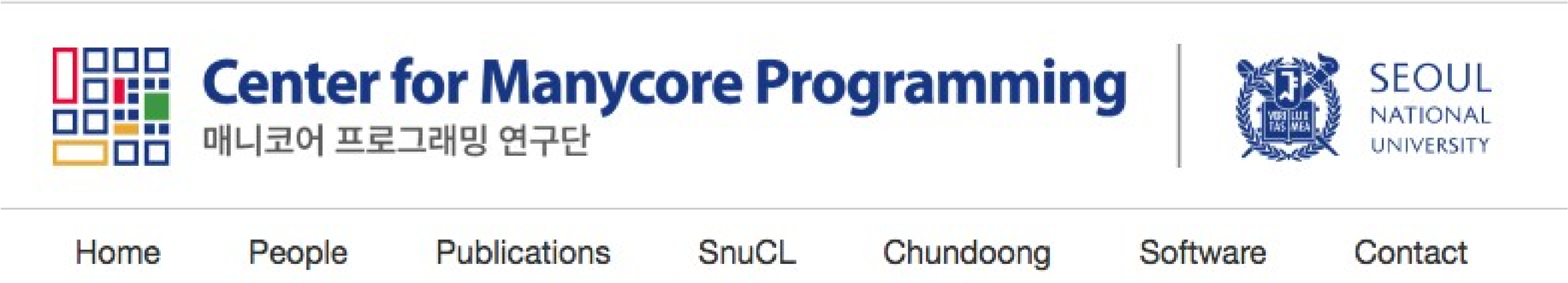
It is an OpenMP program

SP = Pentadiagonal Solver

557.pcsp

It is a C language program





```
//---------------------------------------------------------------------
// FORWARD ELIMINATION
//---------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1,j1,j2)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    j1 = j + 1;
    j2 = j + 2;
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][
      lhsY[3][k][j][i] = fac
      lhsY[4][k][j][i] = fac
      for (m = 0; m < 3; m++
        rhs[m][k][j][i] = fa
      }
      lhsY[2][k][j1][i] = lh                                ][i];
      lhsY[3][k][j1][i] = lh                                ][i];
      for (m = 0; m < 3; m++
        rhs[m][k][j1][i] = r                                [i];
      }
      lhsY[1][k][j2][i] = lh                                ][i];
      lhsY[2][k][j2][i] = lh                                ][i];
      for (m = 0; m < 3; m++
        rhs[m][k][j2][i] = r                                [i];
      }
    }
  }
}
```

4-dimensional loop and

Outer-dimension range: 0, 1, 2

```
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    j1 = j + 1;
    j2 = j + 2;
    for (i = 1; i <= gp0-2; i++) {
          . . .
      for (m = 0; m < 3; m++) {
      }
          . . .
    }
  }
}
```

```
  //-----------------------------------------------------------------------
  // FORWARD ELIMINATION
  //-----------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1,j1,j2)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    j1 = j + 1;
    j2 = j + 2;
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j1][i] = lhsY[2][k][j1][i] - lhsY[1][k][j1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j1][i] = lhsY[3][k][j1][i] - lhsY[1][k][j1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j1][i] = rhs[m][k][j1][i] - lhsY[1][k][j1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j2][i] = lhsY[1][k][j2][i] - lhsY[0][k][j2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j2][i] = lhsY[2][k][j2][i] - lhsY[0][k][j2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j2][i] = rhs[m][k][j2][i] - lhsY[0][k][j2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

i is innermost loop and last coordinate

```
  //------------------------------------------------------------------------
  // FORWARD ELIMINATION
  //------------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1,j1,j2)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    j1 = j + 1;
    j2 = j + 2;
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j1][i] = lhsY[2][k][j1][i] - lhsY[1][k][j1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j1][i] = lhsY[3][k][j1][i] - lhsY[1][k][j1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j1][i] = rhs[m][k][j1][i] - lhsY[1][k][j1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j2][i] = lhsY[1][k][j2][i] - lhsY[0][k][j2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j2][i] = lhsY[2][k][j2][i] - lhsY[0][k][j2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j2][i] = rhs[m][k][j2][i] - lhsY[0][k][j2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

j elements from three rows accessed

data dependence on loop j ⇒ j loop is sequential

```
  //-------------------------------------------------------------------
  // FORWARD ELIMINATION
  //-------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1,j1,j2)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    j1 = j + 1;
    j2 = j + 2;
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j1][i] = lhsY[2][k][j1][i] - lhsY[1][k][j1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j1][i] = lhsY[3][k][j1][i] - lhsY[1][k][j1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j1][i] = rhs[m][k][j1][i] - lhsY[1][k][j1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j2][i] = lhsY[1][k][j2][i] - lhsY[0][k][j2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j2][i] = lhsY[2][k][j2][i] - lhsY[0][k][j2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j2][i] = rhs[m][k][j2][i] - lhsY[0][k][j2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

loop nest is not perfect

Expression Re-materialization

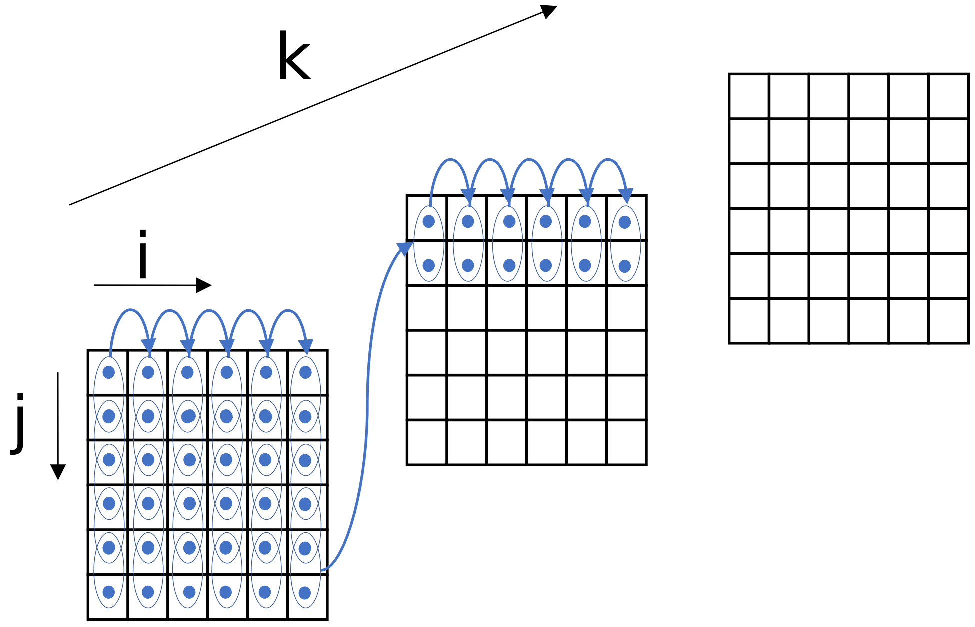
```
  //-------------------------------------------------------------------------
  // FORWARD ELIMINATION
  //-------------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j+1][i] = lhsY[2][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j+1][i] = lhsY[3][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+1][i] = rhs[m][k][j+1][i] - lhsY[1][k][j+1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j+2][i] = lhsY[1][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j+2][i] = lhsY[2][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+2][i] = rhs[m][k][j+2][i] - lhsY[0][k][j+2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

```
  //--------------------------------------------------------------------------
  // FORWARD ELIMINATION
  //--------------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j+1][i] = lhsY[2][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j+1][i] = lhsY[3][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+1][i] = rhs[m][k][j+1][i] - lhsY[1][k][j+1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j+2][i] = lhsY[1][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j+2][i] = lhsY[2][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+2][i] = rhs[m][k][j+2][i] - lhsY[0][k][j+2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

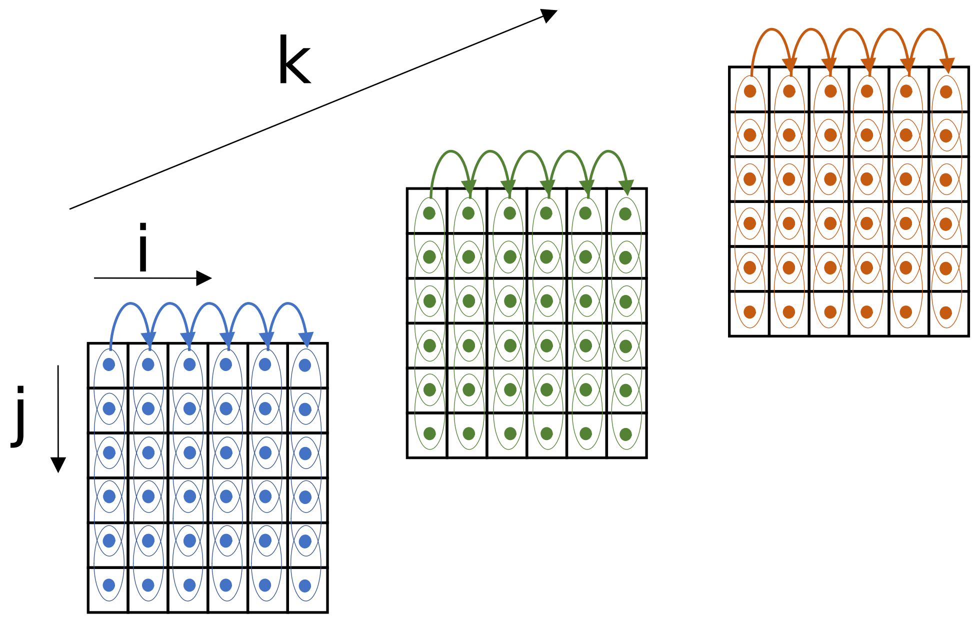
We will focus on m=3

Sequential Execution

```
for (k = 1; k <= gp2-2; k++) {
 for (j = 0; j <= gp1-3; j++) {
    for (i = 1; i <= gp0-2; i++) {
           .    .    .
      lhsY[3][k][j][i] = fac1* lhsY[3][k][j][i];
 .    .    .
   }
  }
}
```

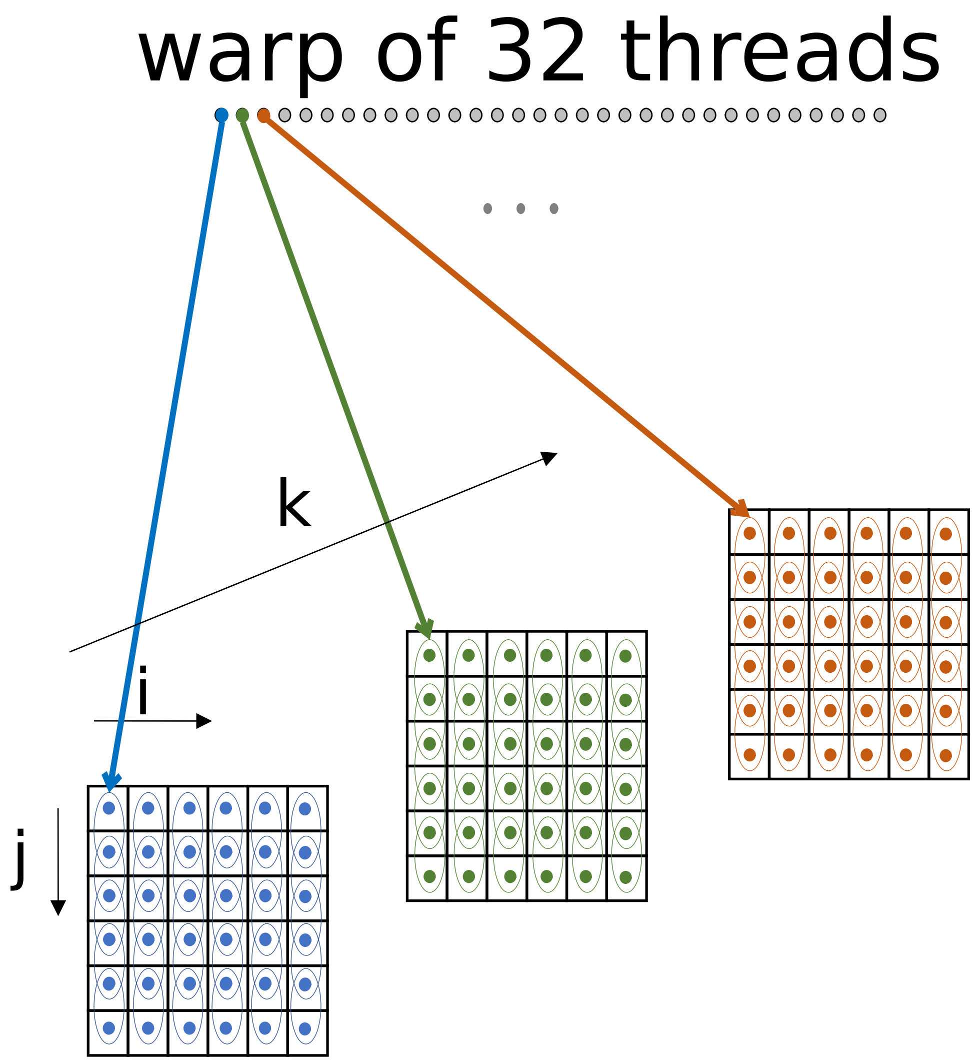


Parallelizing loop k



Intra-thread access pattern.

lhsY[3][k][j][i]



Parallelizing loop k

Inter-thread access pattern?

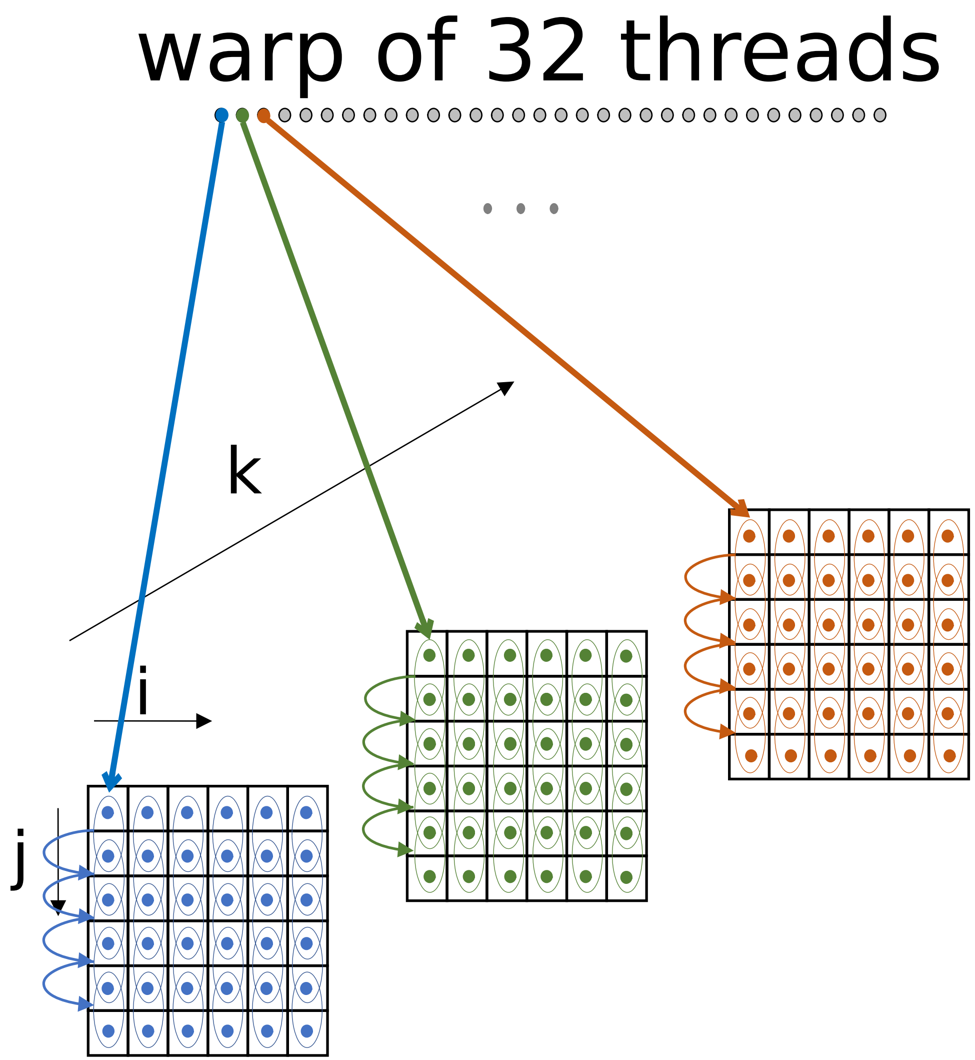
None of the accesses are coalesced

lhsY[3][k][j][i]

```
//------------------------------------------------------------------------
// FORWARD ELIMINATION
//------------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1)
for (k = 1; k <= gp2-2; k++) {
  for (j = 0; j <= gp1-3; j++) {
    for (i = 1; i <= gp0-2; i++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j+1][i] = lhsY[2][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j+1][i] = lhsY[3][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+1][i] = rhs[m][k][j+1][i] - lhsY[1][k][j+1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j+2][i] = lhsY[1][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j+2][i] = lhsY[2][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+2][i] = rhs[m][k][j+2][i] - lhsY[0][k][j+2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

Interchange loops j and i

```
  //-------------------------------------------------------------------
  // FORWARD ELIMINATION
  //-------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1)
for (k = 1; k <= gp2-2; k++) {
  for (i = 0; i <= gp0-2; i++) {
    for (j = 1; j <= gp1-3; j++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j+1][i] = lhsY[2][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j+1][i] = lhsY[3][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+1][i] = rhs[m][k][j+1][i] - lhsY[1][k][j+1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j+2][i] = lhsY[1][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j+2][i] = lhsY[2][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+2][i] = rhs[m][k][j+2][i] - lhsY[0][k][j+2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```



Parallelizing loop k

Inter-thread access pattern?

None of the accesses are coalesced

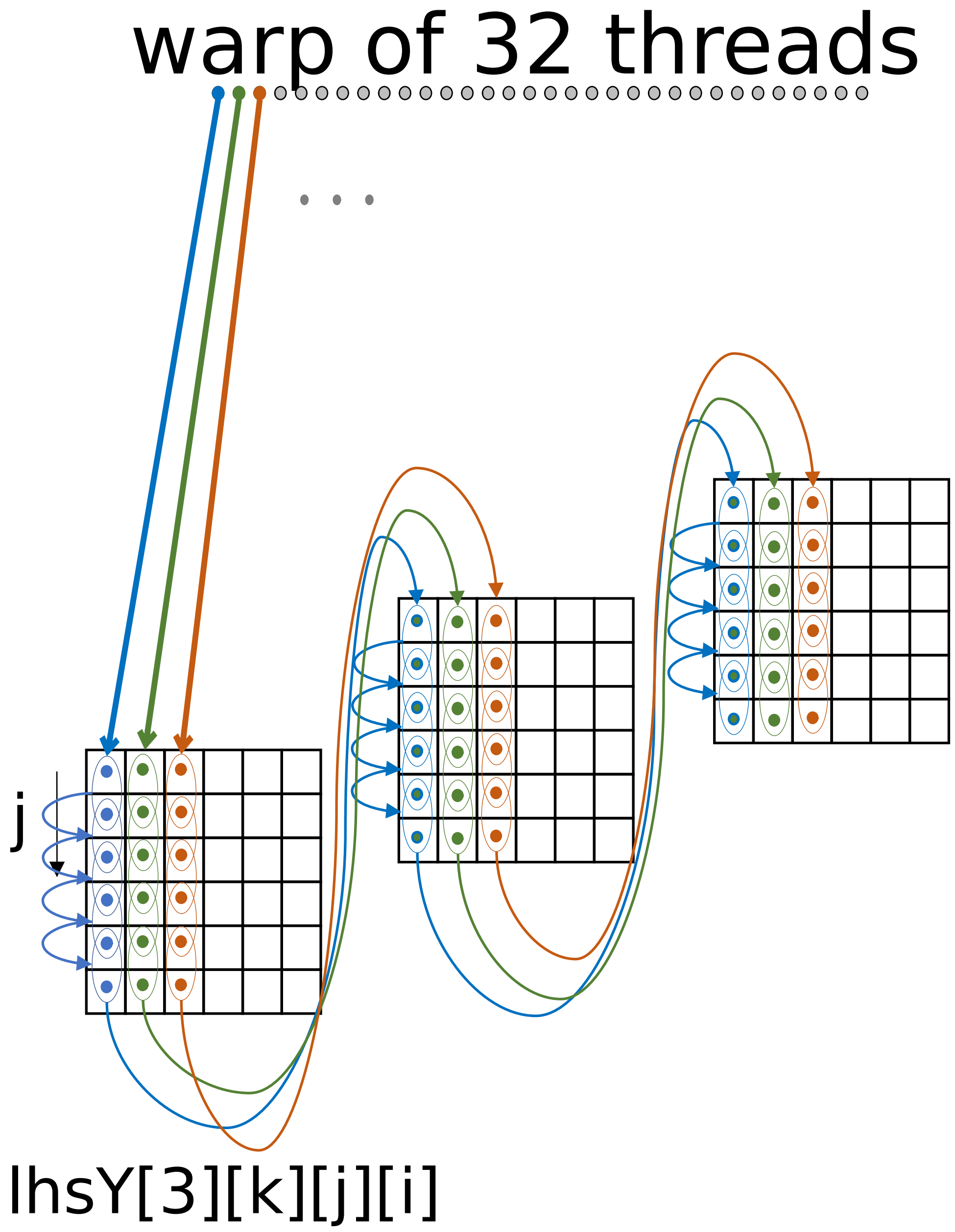
lhsY[3][k][j][i]

```
  //-------------------------------------------------------------------
  // FORWARD ELIMINATION
  //-------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1)
for (k = 1; k <= gp2-2; k++) {
  for (i = 0; i <= gp0-2; i++) {
    for (j = 1; j <= gp1-3; j++) {
      fac1 = 1.0/lhsY[2][k][j][i];
      lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
      lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
      }
      lhsY[2][k][j+1][i] = lhsY[2][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[3][k][j][i];
      lhsY[3][k][j+1][i] = lhsY[3][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+1][i] = rhs[m][k][j+1][i] - lhsY[1][k][j+1][i]*rhs[m][k][j][i];
      }
      lhsY[1][k][j+2][i] = lhsY[1][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[3][k][j][i];
      lhsY[2][k][j+2][i] = lhsY[2][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[4][k][j][i];
      for (m = 0; m < 3; m++) {
        rhs[m][k][j+2][i] = rhs[m][k][j+2][i] - lhsY[0][k][j+2][i]*rhs[m][k][j][i];
      }
    }
  }
}
```

Collapse loops k and i

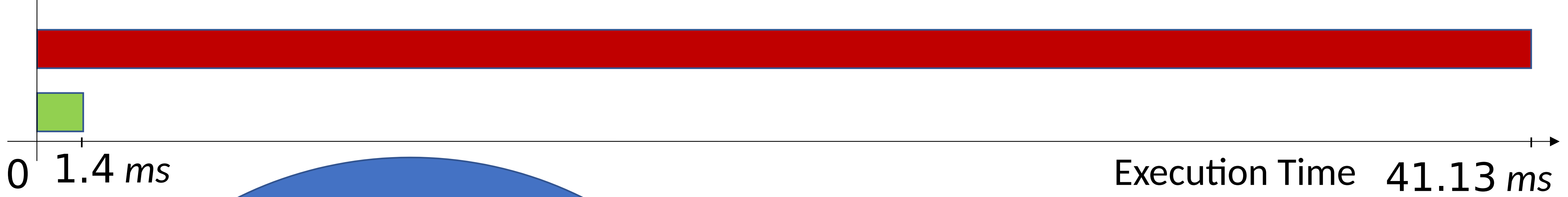
```
  //-------------------------------------------------------------------
  // FORWARD ELIMINATION
  //-------------------------------------------------------------------
#pragma omp target teams distribute parallel for private(i,j,k,m,fac1)
for (int c = 1; c <= ((gp2-2) * gp0-2); ++c) {
  k = c / gp0-2;
  i = c % gp0-2;
  for (j = 1; j <= gp1-3; j++) {
    fac1 = 1.0/lhsY[2][k][j][i];
    lhsY[3][k][j][i] = fac1*lhsY[3][k][j][i];
    lhsY[4][k][j][i] = fac1*lhsY[4][k][j][i];
    for (m = 0; m < 3; m++) {
      rhs[m][k][j][i] = fac1*rhs[m][k][j][i];
    }
    lhsY[2][k][j+1][i] = lhsY[2][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[3][k][j][i];
    lhsY[3][k][j+1][i] = lhsY[3][k][j+1][i] - lhsY[1][k][j+1][i]*lhsY[4][k][j][i];
    for (m = 0; m < 3; m++) {
      rhs[m][k][j+1][i] = rhs[m][k][j+1][i] - lhsY[1][k][j+1][i]*rhs[m][k][j][i];
    }
    lhsY[1][k][j+2][i] = lhsY[1][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[3][k][j][i];
    lhsY[2][k][j+2][i] = lhsY[2][k][j+2][i] - lhsY[0][k][j+2][i]*lhsY[4][k][j][i];
    for (m = 0; m < 3; m++) {
      rhs[m][k][j+2][i] = rhs[m][k][j+2][i] - lhsY[0][k][j+2][i]*rhs[m][k][j][i];
    }
  }
}
```

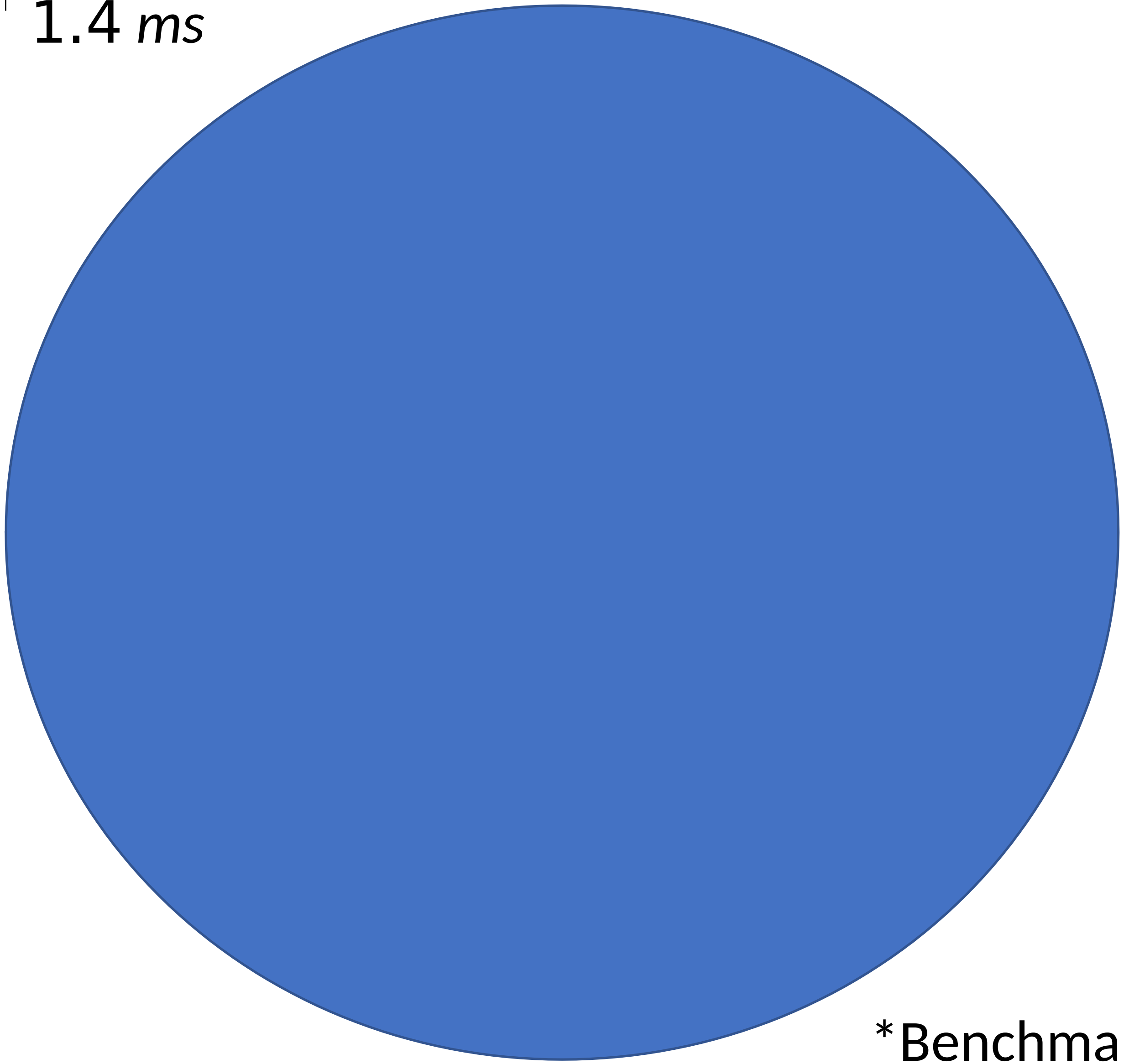
Parallelizing loop c



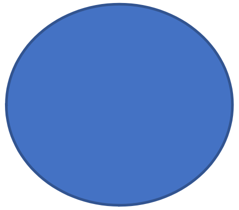
Inter-thread access pattern?

Perfect coalescing





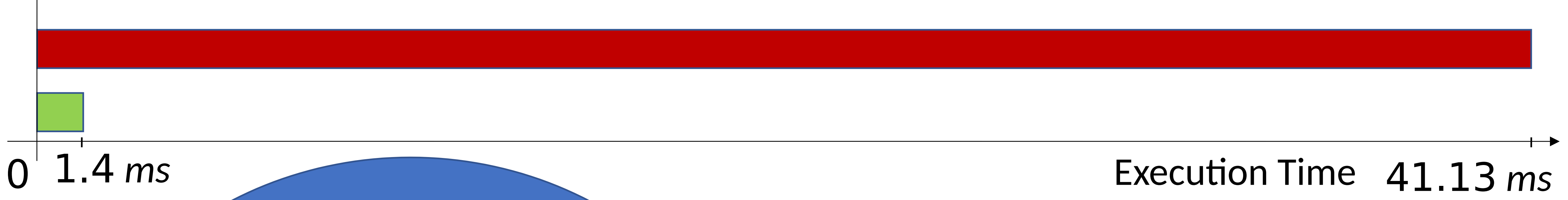
On Nvidia Pascal (P100)
Kernel is 29.4 times faster
After IPAD-enabled transformations

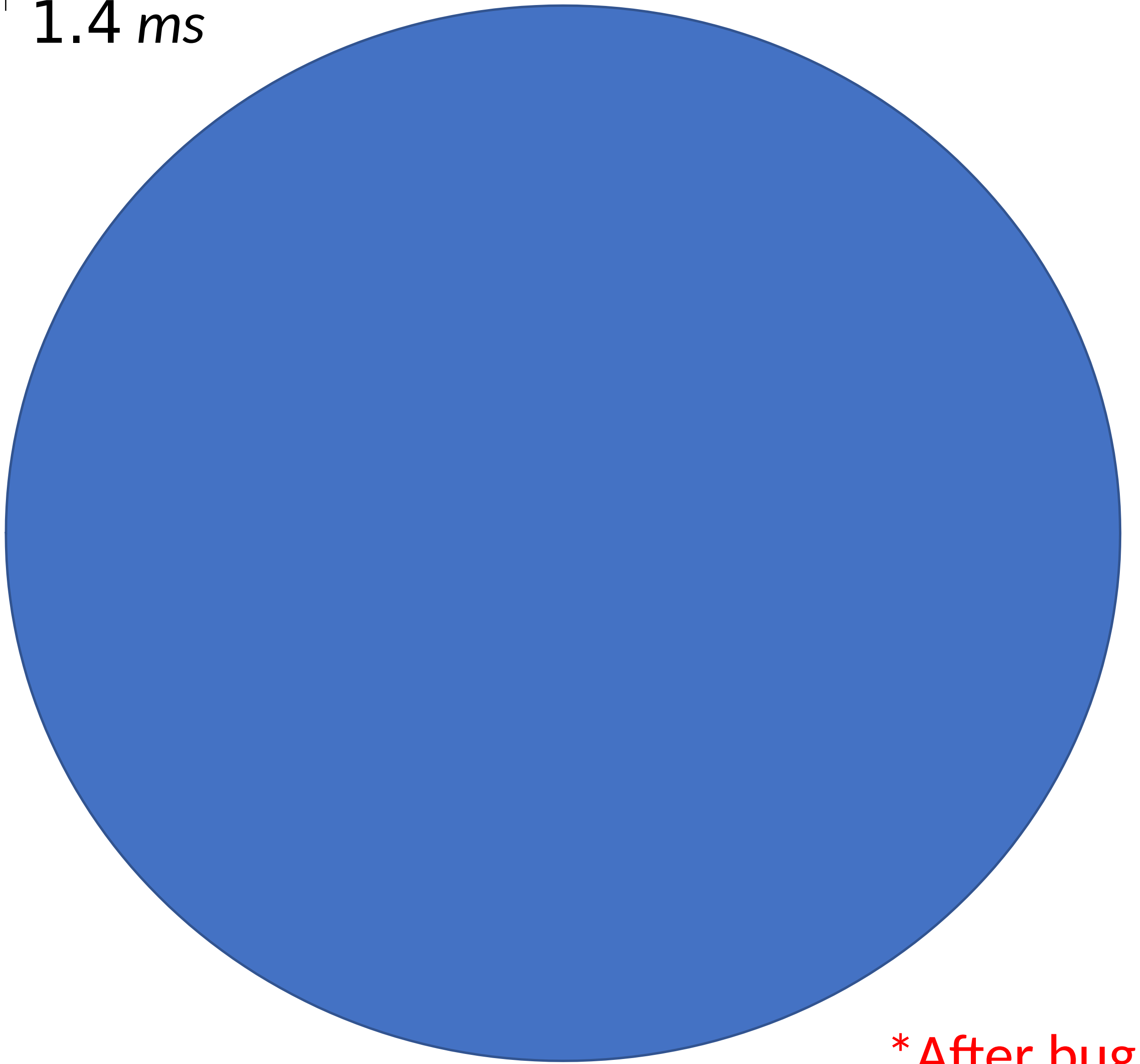


On Nvidia Volta (V100): 16.4 times faster

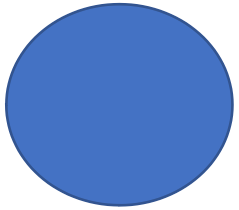
Benchmark speedups:
Pascal (P100): 1.53x
Volta (V100): 1.26x

*Benchmarks was not verifying.





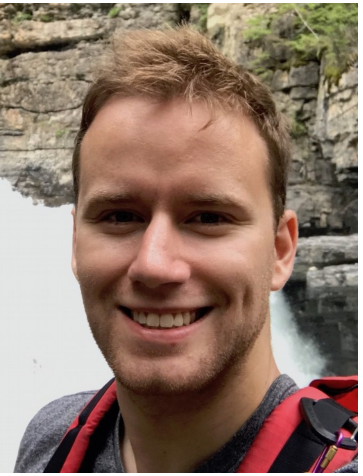
On Nvidia P 88.5 (P100)
Kernel is ~~29.4~~ times faster
After IPAD-enabled transformations



111
On Nvidia Volta (V100): ~~16.4~~ times faster

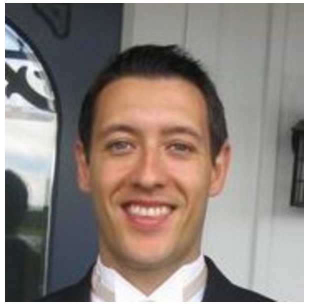
Benchmark speedup
Pascal (P100): ~~1.53x~~ 3.33x
Volta (V100): ~~1.26x~~ 2.3x

*After bug figs with benchmark verifying.

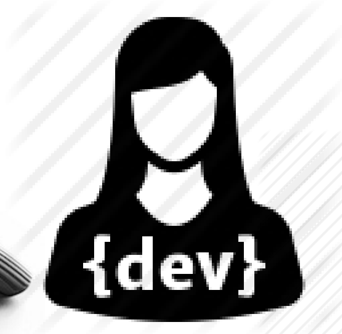


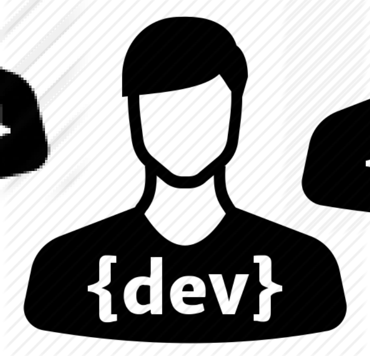
Artem Chikin

J. Nelson Amaral



Ettore Tiotto





October 2017

March 2018

US 20100251210A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2010/0251210 A1**

Amaral et al. (43) **Pub. Date: Sep. 30, 2010**

(54) M V

COMPILER FOR RESTRUCTURING CODE USING ITERATION-POINT ALGEBRAIC DIFFERENCE ANALYSIS

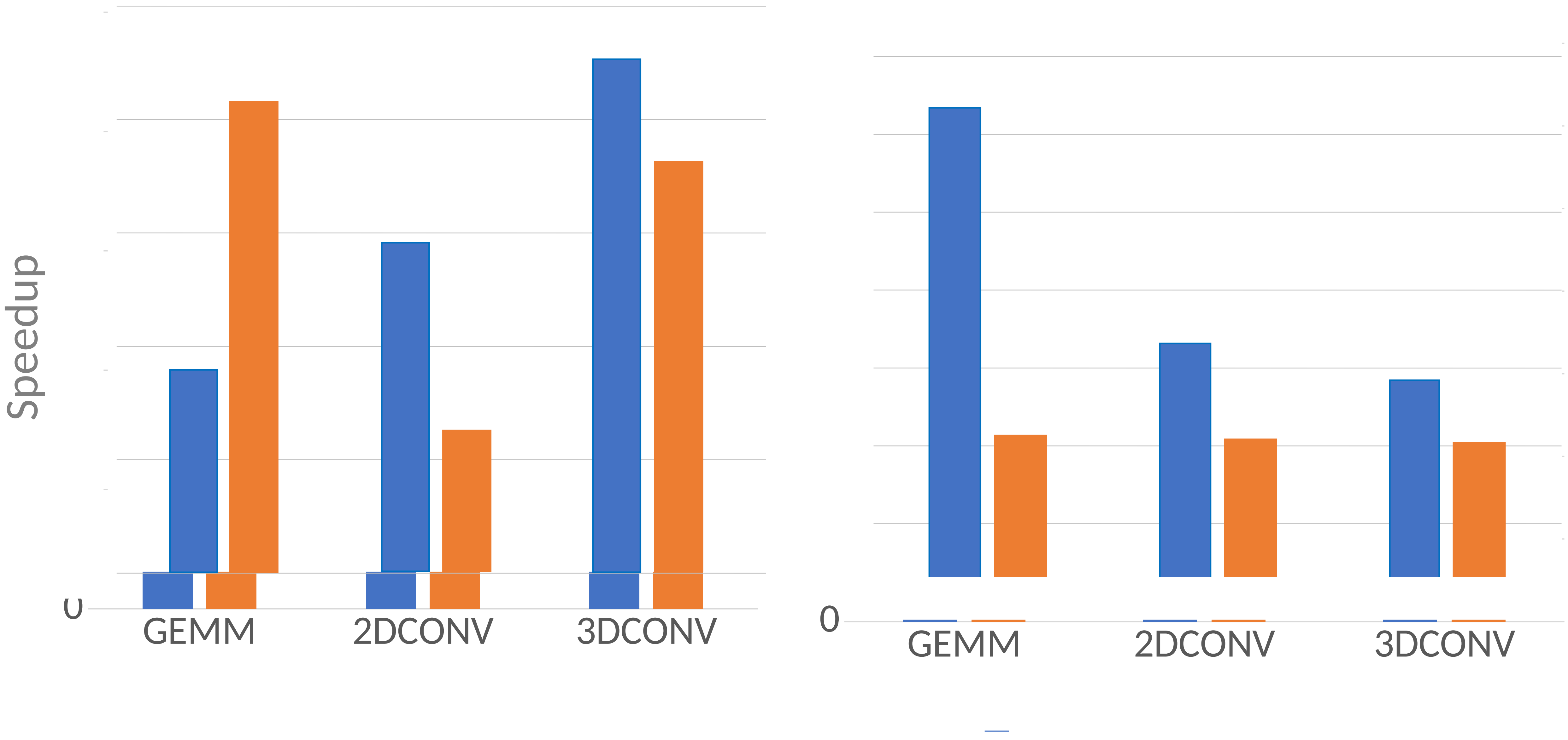
(75) I

3; 717/156

BACKGROUND

[0001] The present invention generally relates to improvements to computer technology and particularly to compilers for improving the efficiency of computer programs by informing restructuring code using iteration-point algebraic difference analysis.

Opportunities in three other Benchmarks


Speedup
0
GEMM
2DCONV
3DCONV
0
GEMM
2DCONV
3DCONV

Symbolic differences of control-dependent expressions …

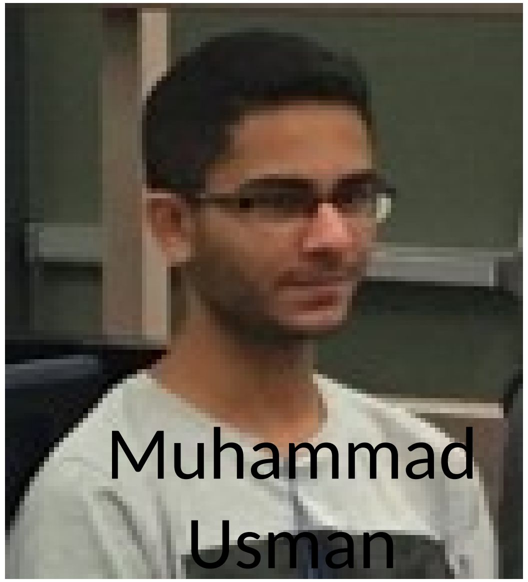
… improve dependence testing …

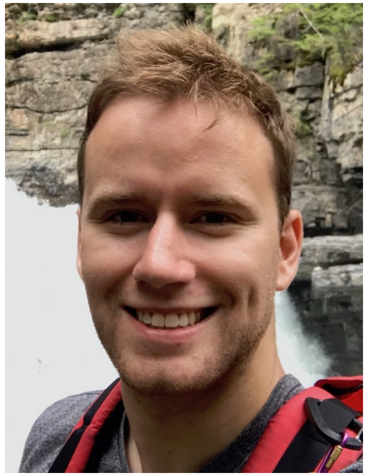
… enable code transformations that were not possible …

… with significant performance improvements …

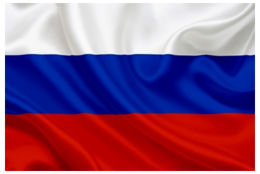
… and enable increased code portability.

May-August 2018





Artem Chikin



OpenMP Target Offloading: Splitting GPU Kernels, Pipelining Communication and Computation, and Selecting Better Grid Geometries

Artem Chikin[1], Tyler Gobran[1], and José Nelson Amaral[1]

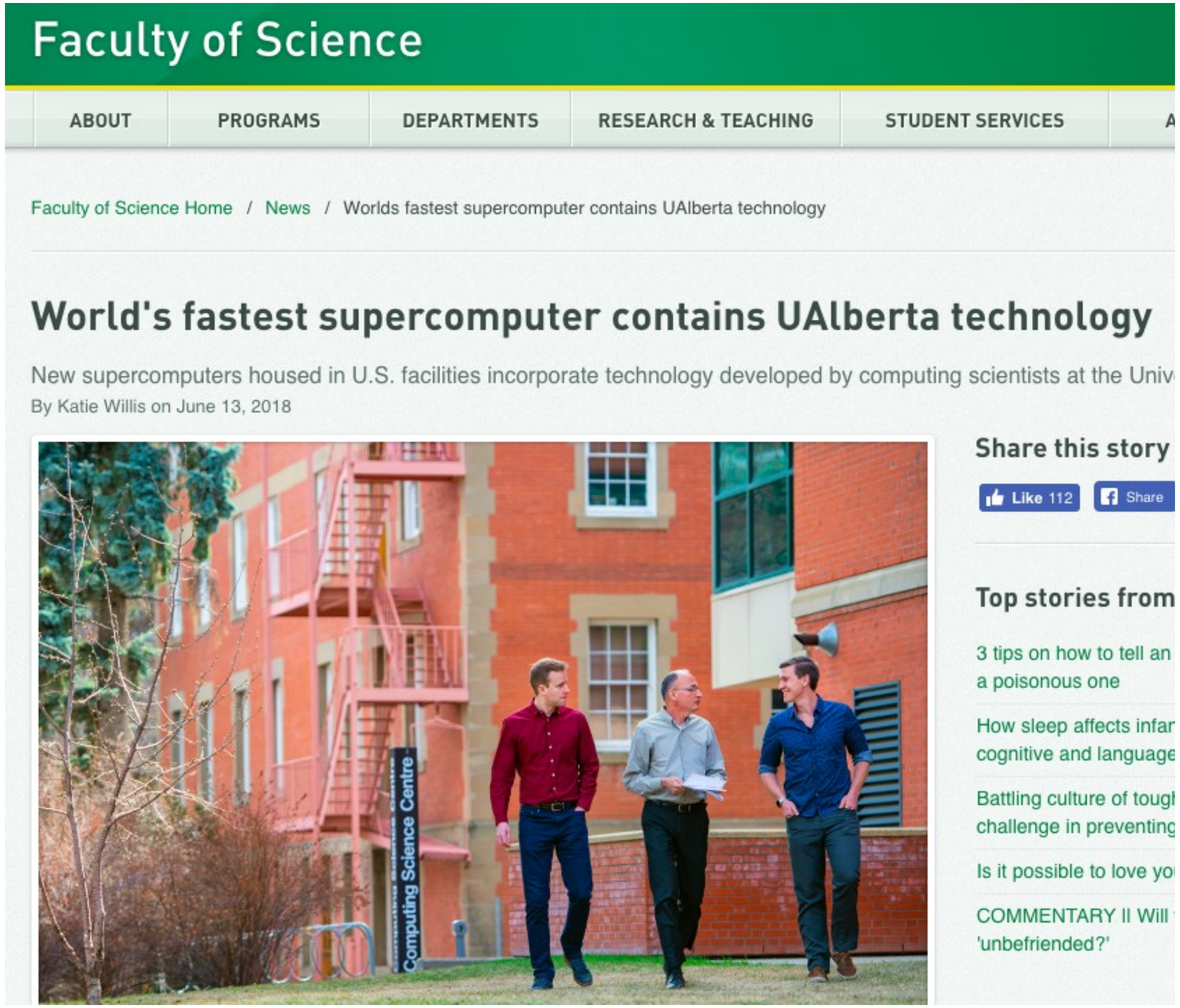
University of Alberta, Edmonton AB, Canac
{artem,gobran,jamaral}@ualberta.ca

SC18
Dallas, TX | hpc inspires.

WACCPD 2018

Fifth Workshop on Accelerator Programming Using Directives

SC18: The International Conference for High Performance Computing, Networking, Storage and Analysis



Faculty of Science

ABOUT | PROGRAMS | DEPARTMENTS | RESEARCH & TEACHING | STUDENT SERVICES

Faculty of Science Home / News / Worlds fastest supercomputer contains UAlberta technology

World's fastest supercomputer contains UAlberta technology

New supercomputers housed in U.S. facilities incorporate technology developed by computing scientists at the Univ

By Katie Willis on June 13, 2018

Share this story

Like 112 Share

Top stories from

3 tips on how to tell an
a poisonous one

How sleep affects infar
cognitive and language

Battling culture of tougl
challenge in preventing

Is it possible to love yo

COMMENTARY II Will
'unbefriended?'

September 2014

July 2018

NVIDIA

Google

Microsoft

IBM

intel

ARM


Artem Chikin


Taylor Lloyd

amazon

moz://a

Google DeepMind

HUAWEI

amazon

codeplay

GitHub