Porting Jacobi on PDC system using OpenMP offloading

Lecture 12

Sunita Chandrasekaran
Associate Professor, University of Delaware
PDC Summer School
Aug 2023

Table of content

- Laplace Serial code – example

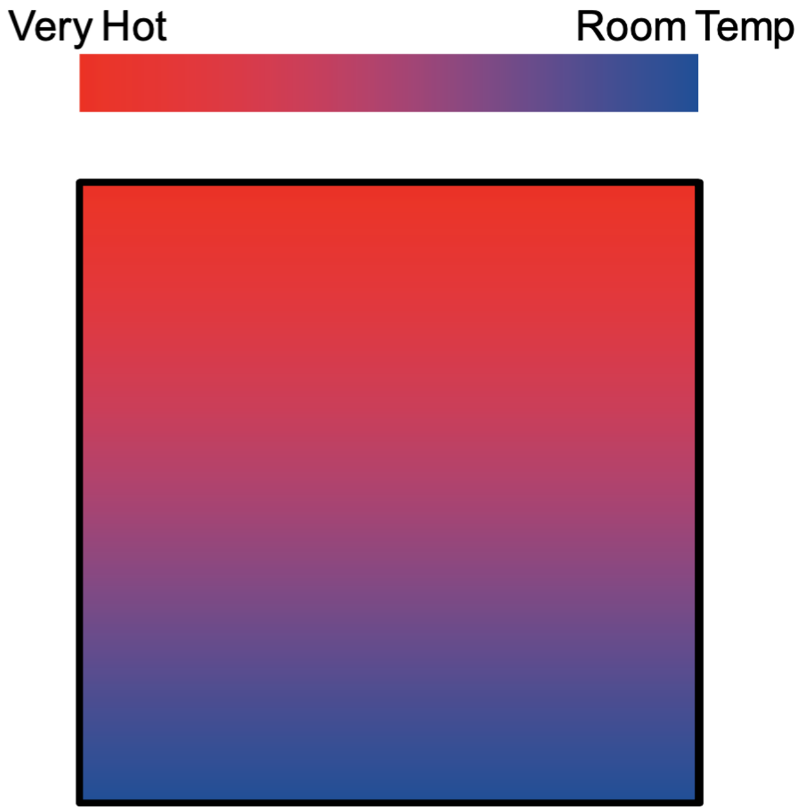
LAPLACE HEAT TRANSFER

Introduction to lab code - visual

We will observe a simple simulation of heat distributing across a metal plate.

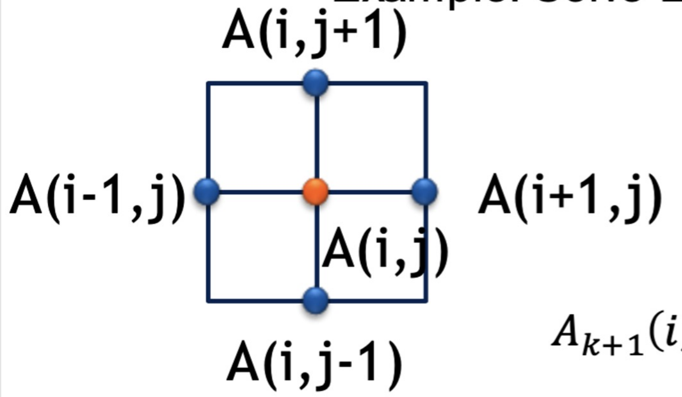
We will apply a consistent heat to the top of the plate.

Then, we will simulate the heat distributing across the plate.



EXAMPLE: JACOBI ITERATION

- Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.
- Common, useful algorithm
- Example: Solve Laplace equation in 2D: $\boldsymbol{\nabla^2 f(x,y) = 0}$



$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

```
while ( error > tol && iter < iter_max )
{
  error = 0.0;
```

Iterate until converged

```
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
    error = fmax( error, fabs(Anew[j][i] - A[j][i]));
  }
}
```

Iterate across matrix elements

Calculate new neighbors

Compute max error for convergence

```
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Swap input/output arrays

Profiling Sequential Code

to use gprof add **-pg** to compile the application

Profile your code to obtain detailed information about how does the code runs:

- Total runtime
- runtime of routines
- Hardware counters

Identify portions that took longer to execute. These are the portions that you will want to parallelize.

LLVM

```
$ amdclang -Ofast -fopenmp -fno-inline -pg -o jacobi-serial
jacobi.c
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 39.672432 s
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.00     24.55    24.55     1000    24.55    24.55  calcNext
  0.00     24.55     0.00     1000     0.00     0.00  swap
  0.00     24.55     0.00        1     0.00     0.00  deallocate
  0.00     24.55     0.00        1     0.00     0.00  initialize
```

gprof misses some of the behaviour by giving 0.00 to swap

Table of content

- Laplace Serial code – example
- Parallelization using target parallel for

```
while ( error > tol && iter < iter_max )
{
  error = 0.0;
```

Parallelize first loop next
OpenMP requires reduction clause

```
//#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
#pragma omp target parallel for map(tofrom: A[:m*n],Anew[:m*n]) \
reduction(max:error)
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
    error = fmax( error, fabs(Anew[j][i] - A[j][i]));
  }
}
```

```
//#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
#pragma omp target parallel for map(tofrom: A[:m*n],Anew[:m*n])
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Parallelize second loop

Build and run the code

- Using PDC
- Module load rocm/5.3.3
- Target which architecture you want to use to compile and execute the code; for example
- amdclang -Ofast -fopenmp --offload-arch=gfx90a -g -o jacobi-omp-llvm-loop jacobi.c
  - **-fopenmp:** tell the compiler that considers openmp pragmas
  - **--offload-arch=gfx90a**: denotes that the target gpu
  - **-Ofast**: an optimization flag that you can add to your compilation command

how to run on PDC

```
salloc --nodes=1 -t 0:30:00 -A pdc-test-2023 -p gpu

amdclang -Ofast -fopenmp -g --offload-arch=gfx90a -o
jacobi-omp-rocm-loop jacobi.c

export LIBOMPTARGET_PROFILE=jacobi-omp-rocm-
loop.json

srun -n 1 ./jacobi-omp-rocm-loop
```

ROCM

```
$ amdclang -Ofast -fopenmp -g --offload-arch=gfx90a -o jacobi-omp-rocm-loop jacobi.c
```

```
salloc --nodes=1 -t 0:30:00 -A pdc-test-2023 -p gpu
```

```
Jacobi relaxation
Calculation: 4096
x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 287.547013
s
```

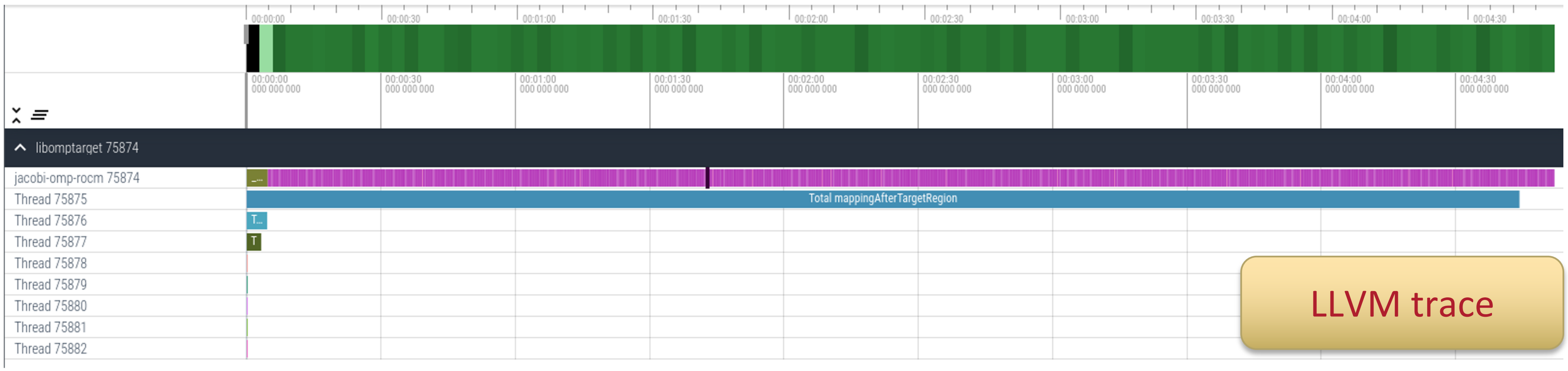
Accelerated code using parallel and no data clauses takes 287.54 on GPUs
about 7 times slower than serial

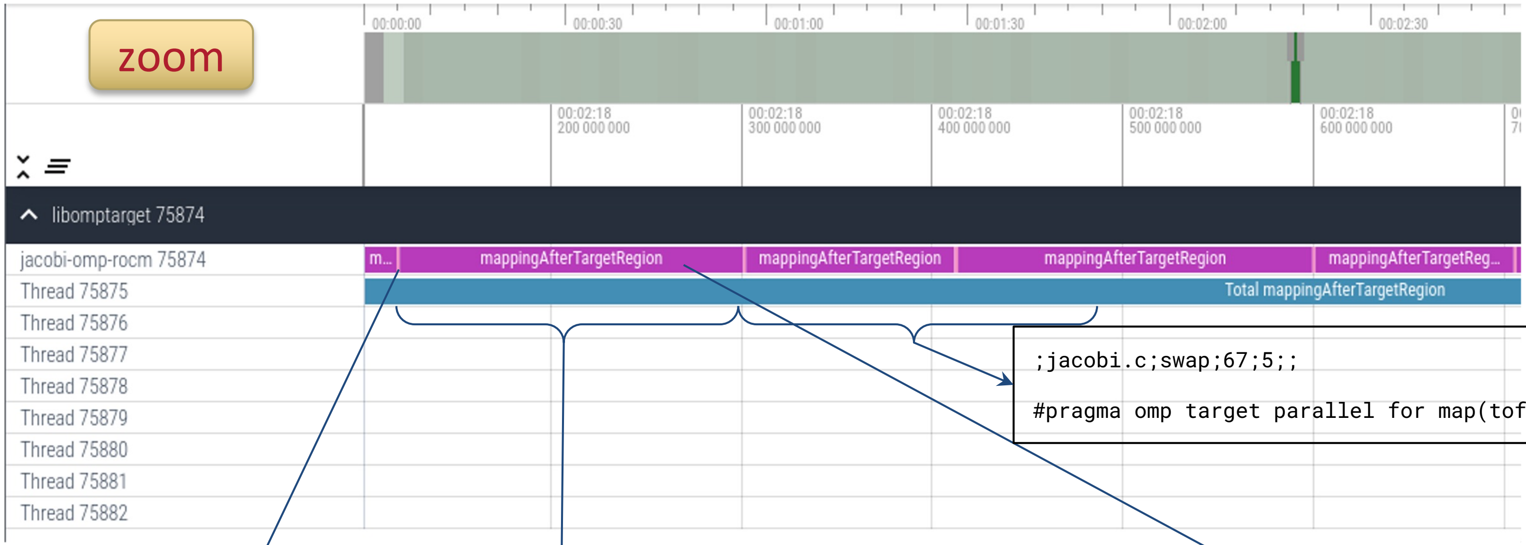
How to trace with LLVM profile on PDC as well as run on PDC

```
salloc --nodes=1 -t 0:30:00 -A pdc-test-2023 -p gpu

export LIBOMPTARGET_PROFILE=prof-jacobi-omp-rocm-loop.json
srun -n 1 ./jacobi-omp-rocm-loop
```

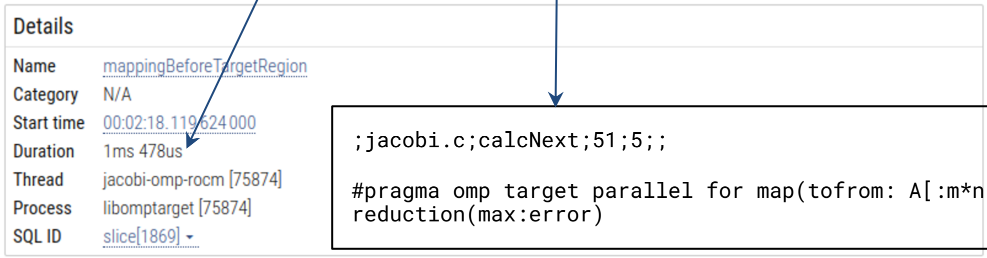
Copy file to your local machine, open browser and open:
https://ui.perfetto.dev/
on the upper left hand “Open trace file”, and select
prof-jacobi-omp-rocm-loop.json to open it

You can zoom in and out with keys “**w**” and “**s**”, and move to
the left and right with “**a**” and “**d**”





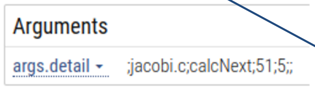
```
;jacobi.c;swap;67;5;;

#pragma omp target parallel for map(tofrom: A[:m*n],Anew[:m*n])
```



Details

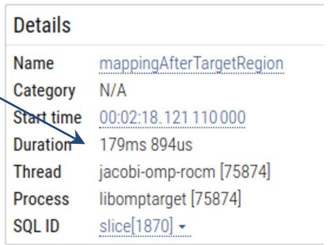
Name	mappingBeforeTargetRegion
Category	N/A
Start time	00:02:18.119 624 000
Duration	1ms 478us
Thread	jacobi-omp-rocm [75874]
Process	libomptarget [75874]
SQL ID	slice[1869]

```
;jacobi.c;calcNext;51;5;;

#pragma omp target parallel for map(tofrom: A[:m*n],Anew[:m*n])
reduction(max:error)
```



Arguments

args.detail	;jacobi.c;calcNext;51;5;;



Details

Name	mappingAfterTargetRegion
Category	N/A
Start time	00:02:18.121 110 000
Duration	179ms 894us
Thread	jacobi-omp-rocm [75874]
Process	libomptarget [75874]
SQL ID	slice[1870]

Helpful information on kernel launch amdclang and LLVM

```
salloc --nodes=1 -t 0:30:00 -A pdc-test-2023 -p gpu
    In case ROCM and LLVM compiler
    (https://openmp.llvm.org/design/Runtimes.html) is used
    export LIBOMPTARGET_INFO=

   for The PDC and amdclang (rocm)
export LIBOMPTARGET_KERNEL_TRACE=1
srun -n 1 ./jacobi
```

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(   1X 256) reqd:(   1X
0) lds_usage:324B sgpr_count:37 vgpr_count:44 sgpr_spill_count:0
vgpr_spill_count:0 tripcount:0 rpc:0
n:__omp_offloading_477a51e4_f801a94c_calcNext_l51

DEVID: 0 SGN:2 ConstWGSize:256  args: 4 teamsXthrds:(   1X 256) reqd:(   1X
0) lds_usage:68B sgpr_count:27 vgpr_count:18 sgpr_spill_count:0
vgpr_spill_count:0 tripcount:0 rpc:0
n:__omp_offloading_477a51e4_f801a94c_swap_l67
```

Improving first openMP version

- The LLVM version is really slow
- Adding **teams distribute**, improves it significantly

```
while ( error > tol && iter < iter_max )
{
  error = 0.0;
```

Parallelize first loop adding teams distribute

OpenMP requires reduction clause

```
//#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
#pragma omp target teams distribute parallel for \
 map(tofrom: A[:m*n],Anew[:m*n]) reduction(max:error)
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
    error = fmax( error, fabs(Anew[j][i] - A[j][i]));
  }
}
```

```
//#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
#pragma omp target teams distribute parallel for \
 map(tofrom: A[:m*n],Anew[:m*n])
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Parallelize second loop

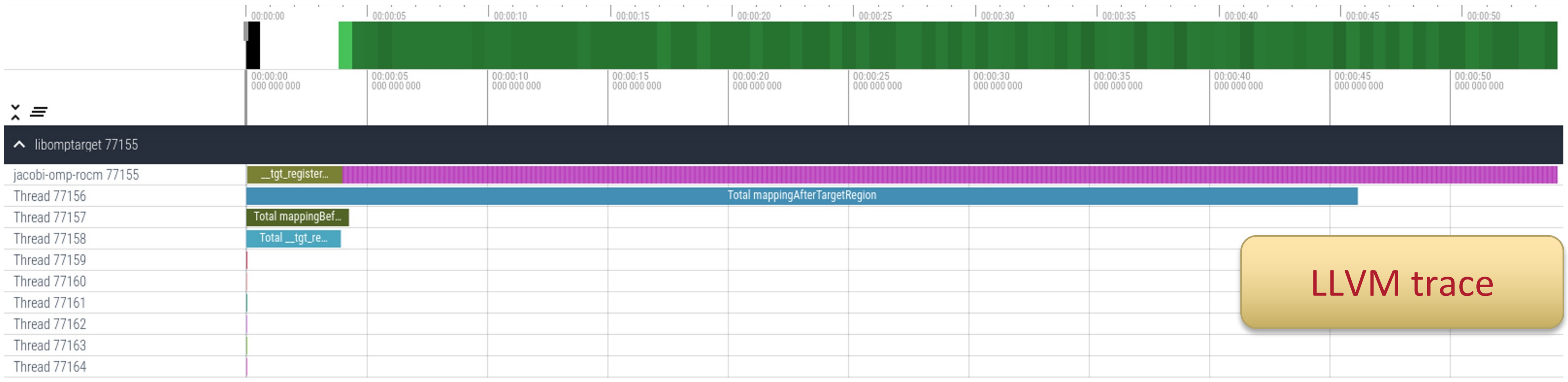
ROCM

```
$ amdclang -Ofast -fopenmp -g --offload-
arch=gfx90a -o jacobi-omp-rocm-loop-teams
jacobi.c
```

```
Jacobi relaxation
Calculation: 4096
x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 49.855616
s
```

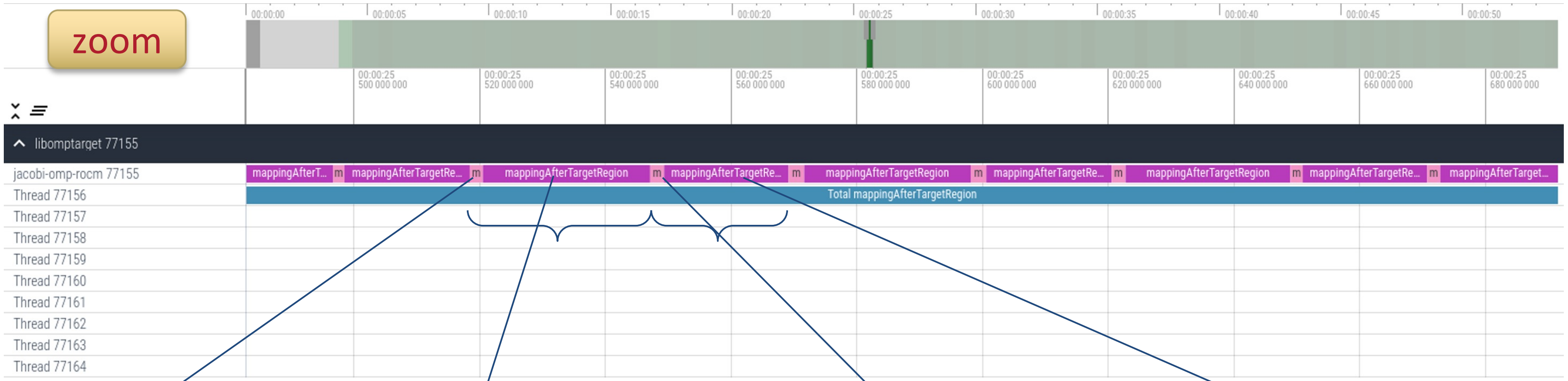
Accelerated code using parallel and no data clauses takes 49.85 on GPUs

still about 1.2 times slower than serial, but 5.7 times faster than the previous version



LLVM trace

zoom



```
;jacobi.c;calcNext;51;5;;
```

```
Duration 2ms 191us          Duration 26ms 562us
```

```
;jacobi.c;swap;67;5;;
```

```
Duration 2ms 259us          Duration 19ms 699us
```

Helpful information on kernel launch amdclang and LLVM

```
salloc --nodes=1 -t 0:30:00 -A pdc-test-2023 -p gpu
```

```
export LIBOMPTARGET_KERNEL_TRACE=1
srun -n 1 ./jacobi
```

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(  16X 256) reqd:(   0X   0)
lds_usage:332B sgpr_count:63 vgpr_count:68 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:4094 rpc:0 n:__omp_offloading_477a51e4_f801a931_calcNext_l51

DEVID: 0 SGN:4 ConstWGSize:256  args: 4 teamsXthrds:(  16X 256) reqd:(   0X   0)
lds_usage:0B sgpr_count:18 vgpr_count:6 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:4094 rpc:0 n:__omp_offloading_477a51e4_f801a931_swap_l67
```

WITHOUT TARGET TEAMS (FIRST IMPLEMENTATION)

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(   1X 256) reqd:(   1X   0)
lds_usage:324B sgpr_count:37 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:0 rpc:0 n:__omp_offloading_477a51e4_f801a94c_calcNext_l51

DEVID: 0 SGN:2 ConstWGSize:256  args: 4 teamsXthrds:(   1X 256) reqd:(   1X   0)
lds_usage:68B sgpr_count:27 vgpr_count:18 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:0 rpc:0 n:__omp_offloading_477a51e4_f801a94c_swap_l67
```

Using LIBOMPTARGET_INFO

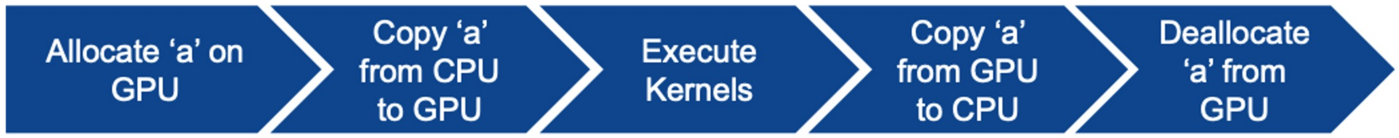
```
export LIBOMPTARGET_INFO=33
srun -n 1 ./jacobi
```

```
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:51:5 with 5 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(error)[8] (implicit)
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007fb5e2ffe010, TgtPtr=0x00007fb5dae00000, Size=134217728,
Name=Anew[:m * n]
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007fb5eafff010, TgtPtr=0x00007fb5caa00000, Size=134217728,
Name=A[:m * n]
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007ffde022a878, TgtPtr=0x00007fb69e600000, Size=8,
Name=error
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007fb69e600000, HstPtr=0x00007ffde022a878, Size=8,
Name=error
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007fb5caa00000, HstPtr=0x00007fb5eafff010, Size=134217728,
Name=A[:m * n]
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007fb5dae00000, HstPtr=0x00007fb5e2ffe010, Size=134217728,
Name=Anew[:m * n]
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:67:5 with 4 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007fb5eafff010, TgtPtr=0x00007fb5dae00000, Size=134217728,
Name=A[:m * n]
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007fb5e2ffe010, TgtPtr=0x00007fb5caa00000, Size=134217728,
Name=Anew[:m * n]
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007fb5caa00000, HstPtr=0x00007fb5e2ffe010, Size=134217728,
Name=Anew[:m * n]
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007fb5dae00000, HstPtr=0x00007fb5eafff010, Size=134217728,
Name=A[:m * n]
```

Table of content

- Laplace Serial code – example
- Parallelization using target parallel for
- Parallelization using target parallel for and data constructs

Our next goal is to add data clauses to our code



```
//#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
#pragma omp target data map(to:A[:m*n],Anew[:m*n])
while ( error > tol && iter < iter_max )
{
   error = 0.0;
```

Create data on the GPUs

```
//#pragma acc parallel loop reduction(max:error) copy(A[:m*n],Anew[:m*n])
#pragma omp target teams distribute parallel for map(tofrom:
A[:m*n],Anew[:m*n]) reduction(max:error)
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
     Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
     error = fmax( error, fabs(Anew[j][i] - A[j][i]));
   }
}
```

Parallelize and max *reduction*

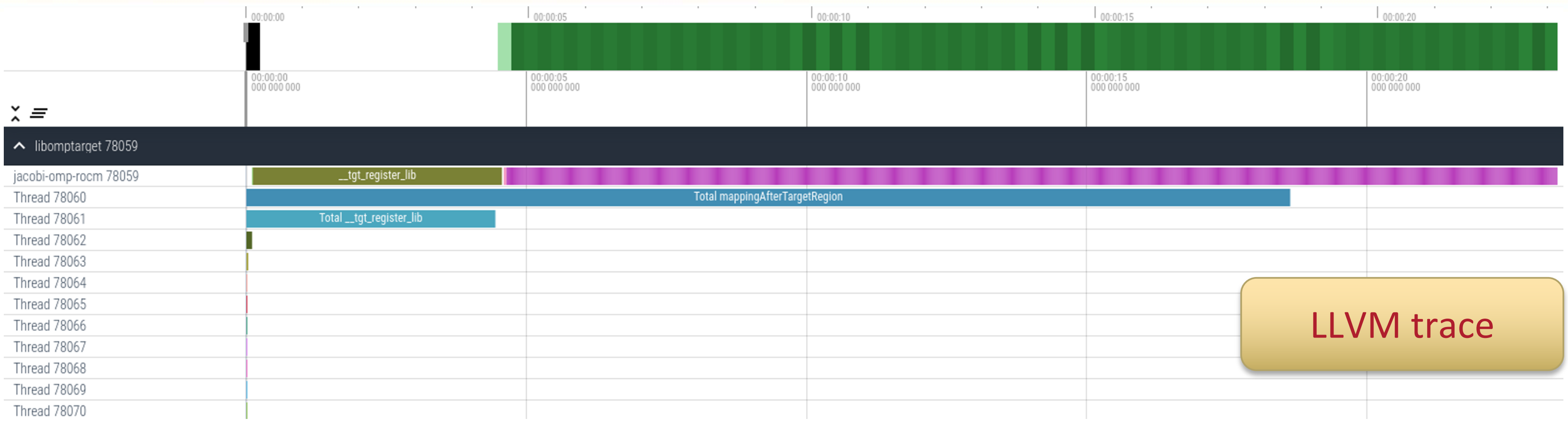
```
//#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
#pragma omp target teams distribute parallel for
map(tofrom: A[:m*n],Anew[:m*n])
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Parallelize second loop

```
$ amdclang -Ofast -fopenmp --offload-
arch=gfx90a -g -o jacobi-omp-rocm-copy
jacobi.c
```

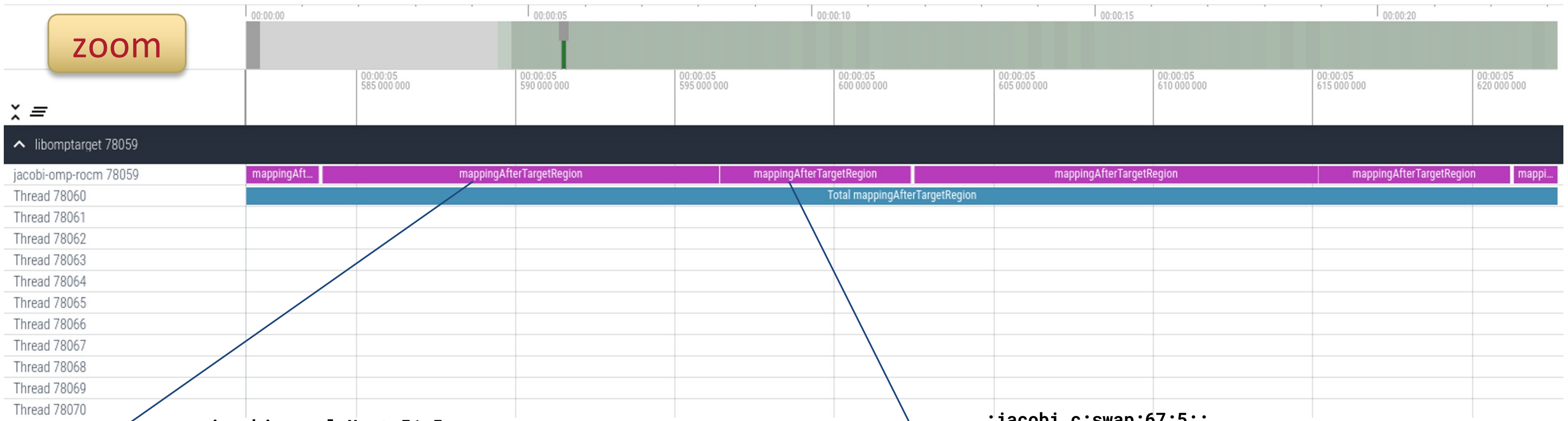
```
Jacobi
relaxation
Calculation:
4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total:
18.736671 s
```

Accelerated code using parallel and data clauses taking 18.7s on GPUs using OpenMP offloading



LLVM trace

zoom



```
;jacobi.c;calcNext;51;5;;
```

```
Duration 12ms 442us
```

```
;jacobi.c;swap;67;5;;
```

```
Duration 5ms 984us
```

This version doesn't have 2 sections, which means doesn't have the copy data to and from the device every kernel execution

Helpful information on kernel launch amdclang and LLVM

```
export LIBOMPTARGET_KERNEL_TRACE=1
srun -n 1 ./jacobi
```

SAME BEHAVIOUR AS THE SECOND (IMPLEMENTATION)

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(  16X 256) reqd:(   0X   0)
lds_usage:332B sgpr_count:63 vgpr_count:68 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:4094 rpc:0 n:__omp_offloading_477a51e4_f801a931_calcNext_l51

DEVID: 0 SGN:4 ConstWGSize:256  args: 4 teamsXthrds:(  16X 256) reqd:(   0X   0)
lds_usage:0B sgpr_count:18 vgpr_count:6 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:4094 rpc:0 n:__omp_offloading_477a51e4_f801a931_swap_l67
```

WHITHOUT TARGET TEAMS (FIRST IMPLEMENTATION)

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(   1X 256) reqd:(   1X   0)
lds_usage:324B sgpr_count:37 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:0 rpc:0 n:__omp_offloading_477a51e4_f801a94c_calcNext_l51

DEVID: 0 SGN:2 ConstWGSize:256  args: 4 teamsXthrds:(   1X 256) reqd:(   1X   0)
lds_usage:68B sgpr_count:27 vgpr_count:18 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:0 rpc:0 n:__omp_offloading_477a51e4_f801a94c_swap_l67
```

Using LIBOMPTARGET_INFO

```
export LIBOMPTARGET_INFO=33
srun -n 1 ./jacobi
```

Improves because there is not memory copy to/from before and after each kernel launch

```
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:51:5 with 5 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(error)[8] (implicit)
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007ffc3bc9d638,
TgtPtr=0x00007f8a28c00000, Size=8, Name=error
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007f8a28c00000,
HstPtr=0x00007ffc3bc9d638, Size=8, Name=error
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:67:5 with 4 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
```

OpenMP collapse

- In order to improve the code, and obtain more parallelism, the collapse() clause

```
//#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
#pragma omp target data map(to:A[:m*n],Anew[:m*n])
while ( error > tol && iter < iter_max )
{
   error = 0.0;
```

Create data on the GPUs

```
//#pragma acc parallel loop reduction(max:error) copy(A[:m*n],Anew[:m*n])
#pragma omp target teams distribute parallel for map(tofrom:
A[:m*n],Anew[:m*n]) reduction(max:error) collapse(2)
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
     Anew[j][i] = 0.25 * ( A[j][i+1] + A[j][i-1] + A[j-1][i] + A[j+1][i]);
     error = fmax( error, fabs(Anew[j][i] - A[j][i]));
   }
}
```

Parallelize and max *reduction*

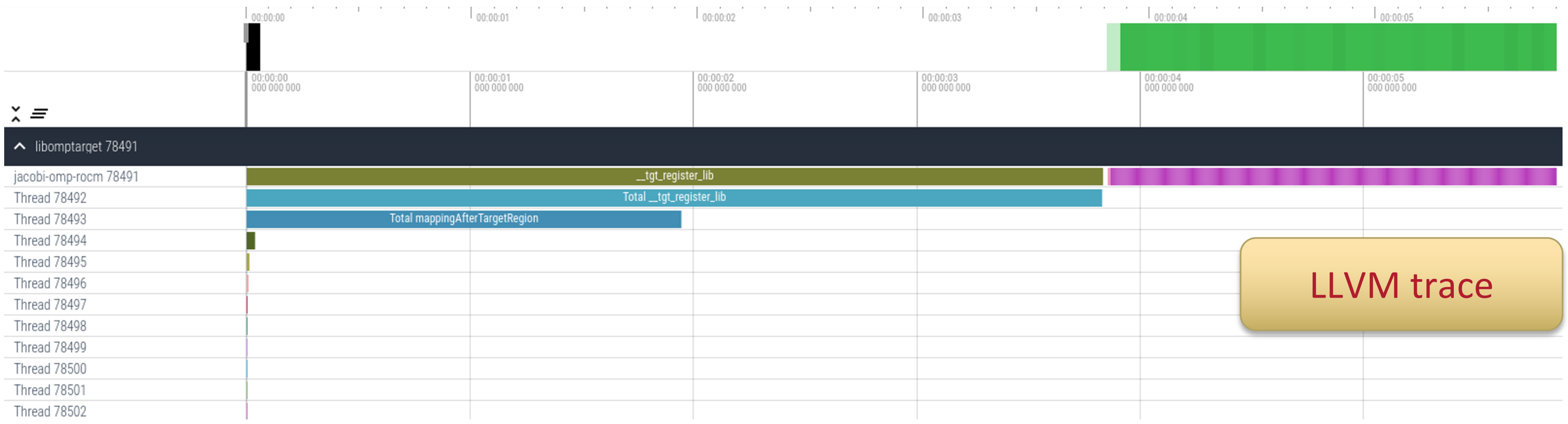
```
//#pragma acc parallel loop copy(A[:m*n],Anew[:m*n])
#pragma omp target teams distribute parallel for
map(tofrom: A[:m*n],Anew[:m*n]) collapse(2)
for( int j = 1; j < n-1; j++)
{
  for( int i = 1; i < m-1; i++ )
  {
    A[j][i] = Anew[j][i];
  }
}
```

Parallelize second loop

```
$ amdclang -Ofast -fopenmp -g --offload-
arch=gfx90a -o jacobi-omp-rocm-copy-
collapse jacobi.c
```

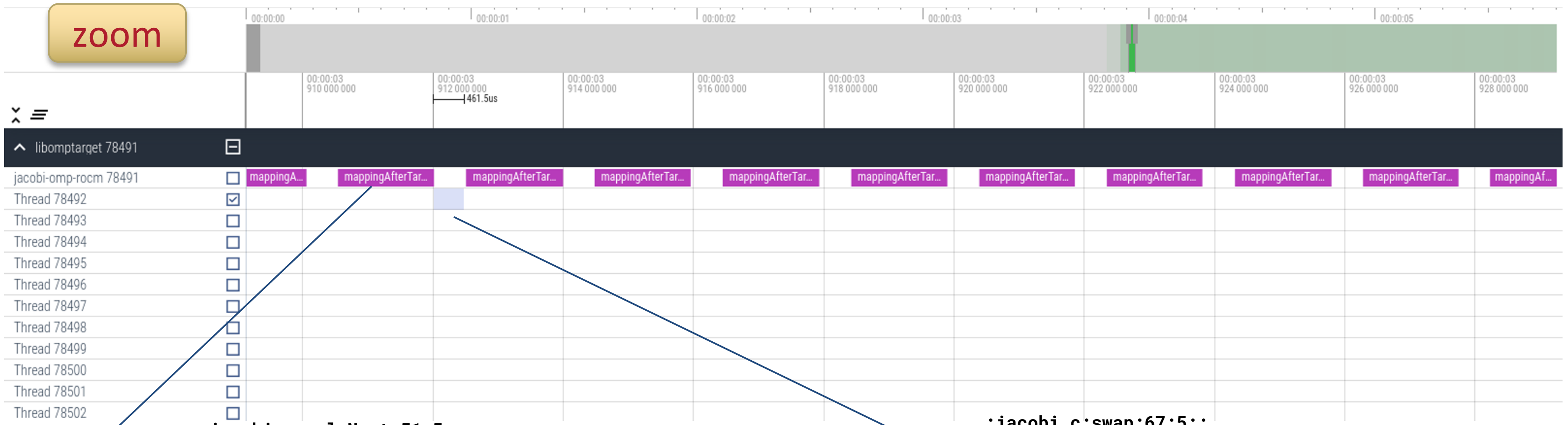
```
Jacobi
relaxation
Calculation:
4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 2.005542
s
```

Accelerated code using parallel and data clauses taking 2.0s on GPUs



LLVM trace

zoom



;jacobi.c;calcNext;51;5;;

```
Duration 1ms 474us
```

;jacobi.c;swap;67;5;;

```
Duration 461us
```

This version doesn’t show the swap kernel most likely because it is less than 1 ms

Helpful information on kernel launch amdclang and LLVM

```
export LIBOMPTARGET_KERNEL_TRACE=1
srun -n 1 ./jacobi
```

COLLAPSE VERSION

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:( 440X 256) reqd:(   0X   0)
lds_usage:332B sgpr_count:61 vgpr_count:60 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:16760836 rpc:0 n:__omp_offloading_477a51e4_f801a942_calcNext_l51
DEVID: 0 SGN:2 ConstWGSize:256  args: 4 teamsXthrds:( 440X 256) reqd:(   0X   0)
lds_usage:68B sgpr_count:31 vgpr_count:29 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:16760836 rpc:0 n:__omp_offloading_477a51e4_f801a942_swap_l67
```

SECOND VERSION (NO COLLAPSE)

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(  16X 256) reqd:(   0X   0)
lds_usage:332B sgpr_count:63 vgpr_count:68 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:4094 rpc:0 n:__omp_offloading_477a51e4_f801a931_calcNext_l51

DEVID: 0 SGN:4 ConstWGSize:256  args: 4 teamsXthrds:(  16X 256) reqd:(   0X   0)
lds_usage:0B sgpr_count:18 vgpr_count:6 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:4094 rpc:0 n:__omp_offloading_477a51e4_f801a931_swap_l67
```

WITHOUT TARGET TEAMS (FIRST IMPLEMENTATION)

```
DEVID: 0 SGN:2 ConstWGSize:256  args: 5 teamsXthrds:(   1X 256) reqd:(   1X   0)
lds_usage:324B sgpr_count:37 vgpr_count:44 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:0 rpc:0 n:__omp_offloading_477a51e4_f801a94c_calcNext_l51


DEVID: 0 SGN:2 ConstWGSize:256  args: 4 teamsXthrds:(   1X 256) reqd:(   1X   0)
lds_usage:68B sgpr_count:27 vgpr_count:18 sgpr_spill_count:0 vgpr_spill_count:0
tripcount:0 rpc:0 n:__omp_offloading_477a51e4_f801a94c_swap_l67
```

Using LIBOMPTARGET_INFO

```
export LIBOMPTARGET_INFO=33
srun -n 1 ./jacobi
```

Same as previous, respect memory copy

```
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:51:5 with 5 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(error)[8] (implicit)
Libomptarget device 0 info: Copying data from host to device, HstPtr=0x00007ffda7425648, TgtPtr=0x00007f13ba800000,
Size=8, Name=error
Libomptarget device 0 info: Copying data from device to host, TgtPtr=0x00007f13ba800000, HstPtr=0x00007ffda7425648,
Size=8, Name=error
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:67:5 with 4 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
Libomptarget device 0 info: Entering OpenMP kernel at jacobi.c:51:5 with 5 arguments:
Libomptarget device 0 info: firstprivate(n)[4] (implicit)
Libomptarget device 0 info: firstprivate(m)[4] (implicit)
Libomptarget device 0 info: tofrom(Anew[:m * n])[134217728]
Libomptarget device 0 info: tofrom(A[:m * n])[134217728]
Libomptarget device 0 info: tofrom(error)[8] (implicit)
```

OpenMP running on CPU

- Instead of offload to GPU we can offload to CPUs multicore

Using LLVM and AMD MI250x

Use multicore flag

Set cores

```
$ clang -Ofast -g -fopenmp --target=x86_64-pc-linux-gnu -o
```

```
$ export OMP_NUM_THREADS=128
$ srun -n 1 jacobi-omp-llvm-host jacobi.c
Jacobi relaxation Calculation: 4096 x 4096 mesh
    0, 0.250000
  100, 0.002397
  200, 0.001204
  300, 0.000804
  400, 0.000603
  500, 0.000483
  600, 0.000403
  700, 0.000345
  800, 0.000302
  900, 0.000269
 total: 3.039321 s
```

Parallelized code using parallel construct took 3.03s on 128 core CPU

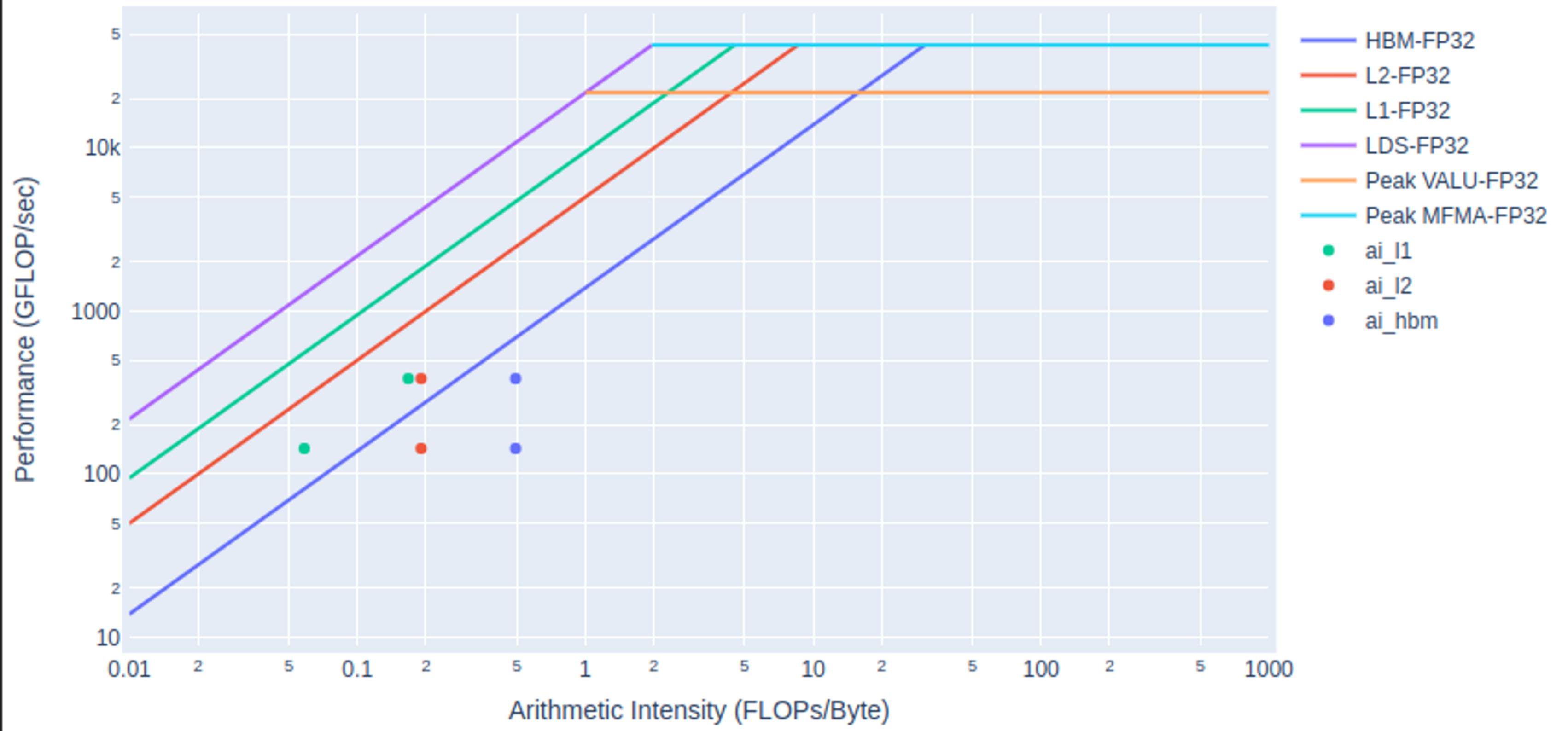
AMD EPYC 7763 64-Core Processor

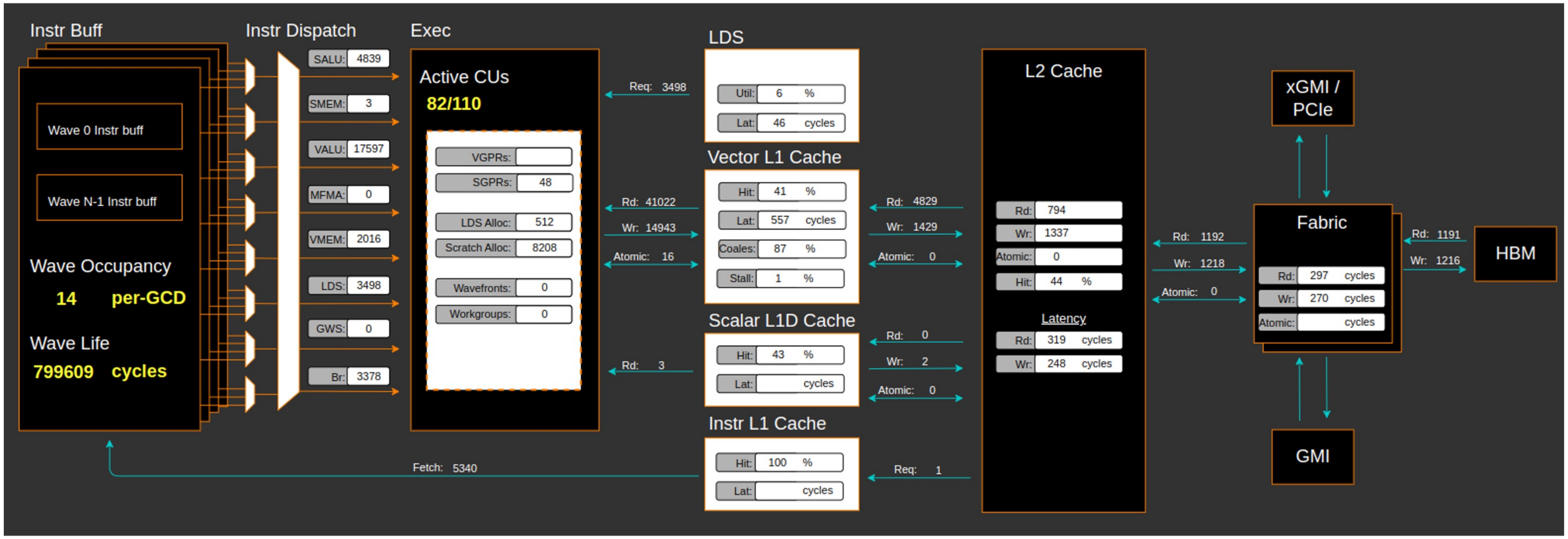
Code Versions	OpenMP			OpenACC
	Perlmutter Nvidia GPU		PDC AMD GPU	
	nvc	llvm	amdclang (rocm)	
Serial	23.364053s	25.557923	39.672432	23.364053s
parallel for	89.513495	242.194770	287.547013	84.040213
Teams distribute	89.992197	100.463713	49.855616	
copy teams parallel for	6.215937	3.666144	18.736671	1.589625
copy teams parallel for collapse	1.604023	1.222854	2.005542	
multicore	2.269870	2.277123	3.039321	0.852060

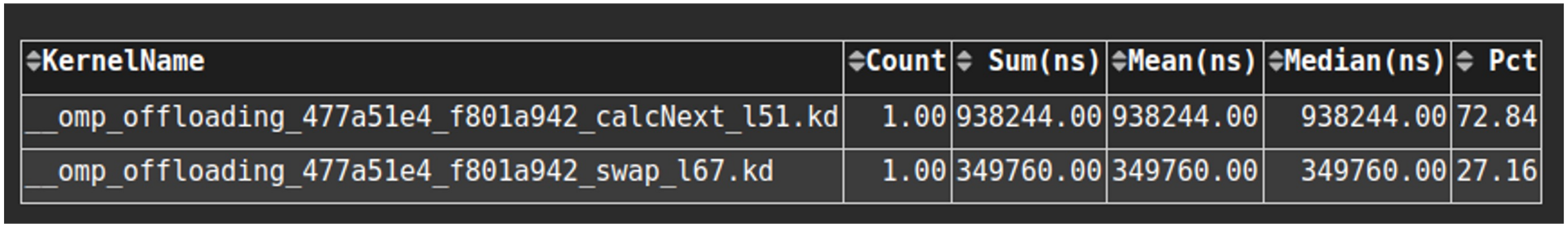
Installing omniperf on PDC

```
module load PDC/22.06
module load cray-python/3.9.12.1
ml easybuild-user/4.6.2
eb CMake-3.22.1.eb
module load cmake/3.23.0

https://amdresearch.github.io/omniperf/installation.html
wget
https://github.com/AMDResearch/omniperf/releases/download/v1.0.8/omniperf-
v1.0.8.tar.gz
tar xfz omniperf-v1.0.8.tar.gz
cd omniperf-1.0.8
mkdir ~/Omniperf
export INSTALL_DIR=~/Omniperf
python3 -m pip install -t ${INSTALL_DIR}/python-libs -r requirements.txt
#SOme error are generated
#ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the
following dependency conflicts.
#dask 2021.10.0 requires cloudpickle>=1.1.1, which is not installed.
#scipy 1.6.2 requires numpy<1.23.0,>=1.16.5, but you have numpy 1.25.1
which is incompatible.

mkdir build
```

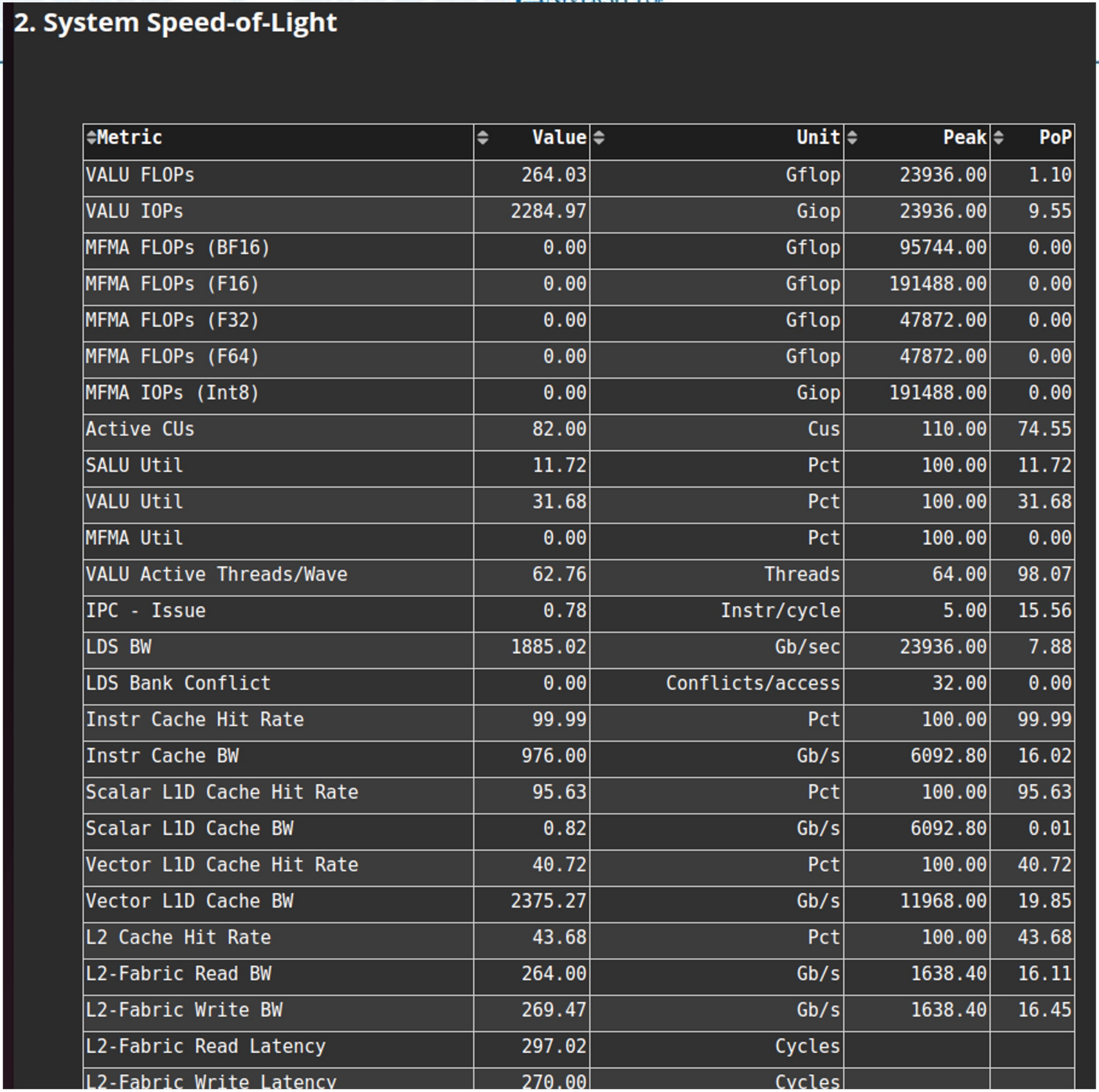
Empirical Roofline Analysis (FP32/FP64)







KernelName	Count	Sum(ns)	Mean(ns)	Median(ns)	Pct
__omp_offloading_477a51e4_f801a942_calcNext_l51.kd	1.00	938244.00	938244.00	938244.00	72.84
__omp_offloading_477a51e4_f801a942_swap_l67.kd	1.00	349760.00	349760.00	349760.00	27.16



2. System Speed-of-Light

Metric	Value	Unit	Peak	PoP
VALU FLOPs	264.03	Gflop	23936.00	1.10
VALU IOPs	2284.97	Giop	23936.00	9.55
MFMA FLOPs (BF16)	0.00	Gflop	95744.00	0.00
MFMA FLOPs (F16)	0.00	Gflop	191488.00	0.00
MFMA FLOPs (F32)	0.00	Gflop	47872.00	0.00
MFMA FLOPs (F64)	0.00	Gflop	47872.00	0.00
MFMA IOPs (Int8)	0.00	Giop	191488.00	0.00
Active CUs	82.00	Cus	110.00	74.55
SALU Util	11.72	Pct	100.00	11.72
VALU Util	31.68	Pct	100.00	31.68
MFMA Util	0.00	Pct	100.00	0.00
VALU Active Threads/Wave	62.76	Threads	64.00	98.07
IPC - Issue	0.78	Instr/cycle	5.00	15.56
LDS BW	1885.02	Gb/sec	23936.00	7.88
LDS Bank Conflict	0.00	Conflicts/access	32.00	0.00
Instr Cache Hit Rate	99.99	Pct	100.00	99.99
Instr Cache BW	976.00	Gb/s	6092.80	16.02
Scalar L1D Cache Hit Rate	95.63	Pct	100.00	95.63
Scalar L1D Cache BW	0.82	Gb/s	6092.80	0.01
Vector L1D Cache Hit Rate	40.72	Pct	100.00	40.72
Vector L1D Cache BW	2375.27	Gb/s	11968.00	19.85
L2 Cache Hit Rate	43.68	Pct	100.00	43.68
L2-Fabric Read BW	264.00	Gb/s	1638.40	16.11
L2-Fabric Write BW	269.47	Gb/s	1638.40	16.45
L2-Fabric Read Latency	297.02	Cycles		
L2-Fabric Write Latency	270.00	Cycles		

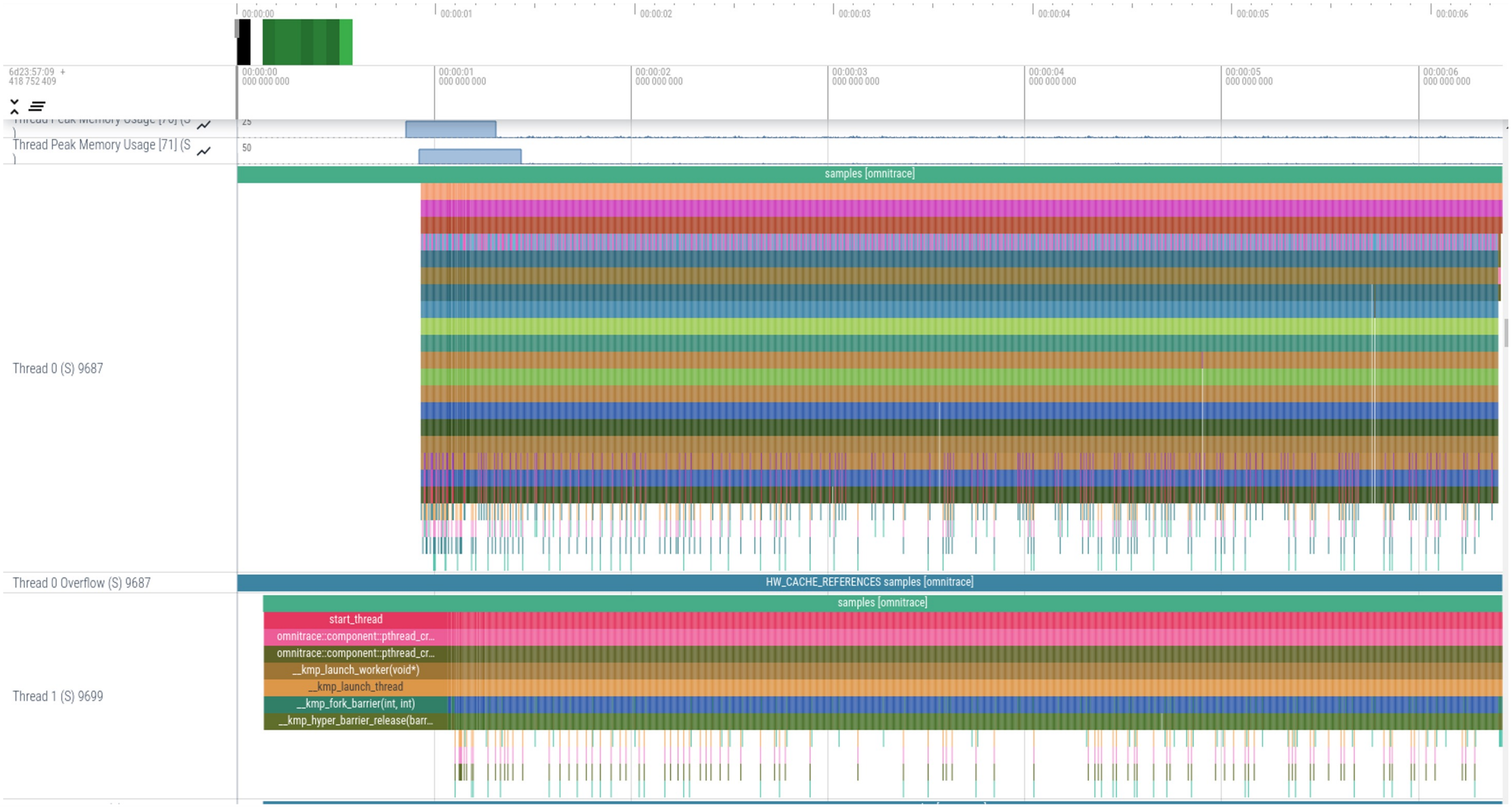
5. Command Processor (CPC/CPF)
6. Shader Processor Input (SPI)
7. Wavefront
10. Compute Units - Instruction Mix
11. Compute Units - Compute Pipeline
12. Local Data Share (LDS)
13. Instruction Cache
14. Scalar L1 Data Cache
15. Texture Addresser and Texture Data (TA/TD)
16. Vector L1 Data Cache
17. L2 Cache
18. L2 Cache (per Channel)

Installing omnitrace

```
module load rocm/5.3.3
export OMNITRACE_VERSION=latest
export ROCM_VERSION=5.3.3
export OMNITRACE_INSTALL_DIR=~/Omnitrace
wget https://github.com/AMDResearch/omnitrace/releases/download/v1.10.1/omnitrace-install.py
python3 omnitrace-install.py -p ~/Omnitrace --rocm 5.3.3


Set up environment:
source ~/Omnitrace/share/omnitrace/setup-env.shOpenMP running on CPU
```

```
omnitrace-sample -- ./jacobi-omp-rocm-loop
```

under the omnitrace-jacobi-omp-rocm-loop-output/ directory, find the .proto file and download it to your local machine, then, you can open it using https://ui.perfetto.dev/


00:00:00
00:00:01
00:00:02
00:00:03
00:00:04
00:00:05
00:00:06
6d23:57:09 + 418 752 409
Thread Peak Memory Usage [70] (S)
Thread Peak Memory Usage [71] (S)
25
50
samples [omnitrace]
Thread 0 (S) 9687
Thread 0 Overflow (S) 9687
HW_CACHE_REFERENCES samples [omnitrace]
Thread 1 (S) 9699
start_thread
omnitrace::component::pthread_cr...
__kmp_launch_worker(void*)
__kmp_launch_thread
__kmp_fork_barrier(int, int)
__kmp_hyper_barrier_release(barr...