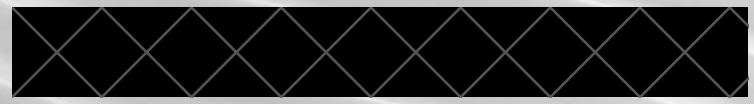
Hewlett Packard
Enterprise

Preparing an application for Hybrid Supercomuting

John Levesque, Harvey Richardson, Alfio Lazzaro,
Nina Mujkanovic, HPE

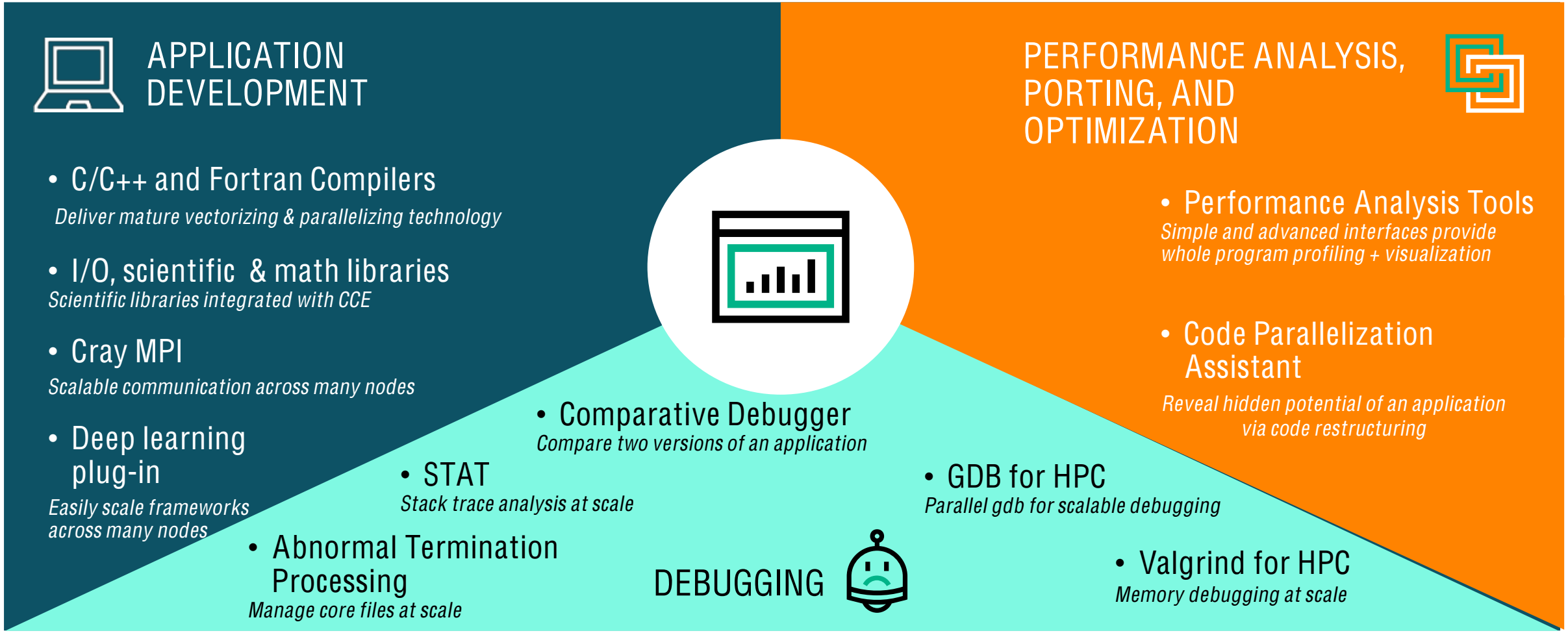


Using HPE Programming Environment for GPUS - DEC 2023

Agenda

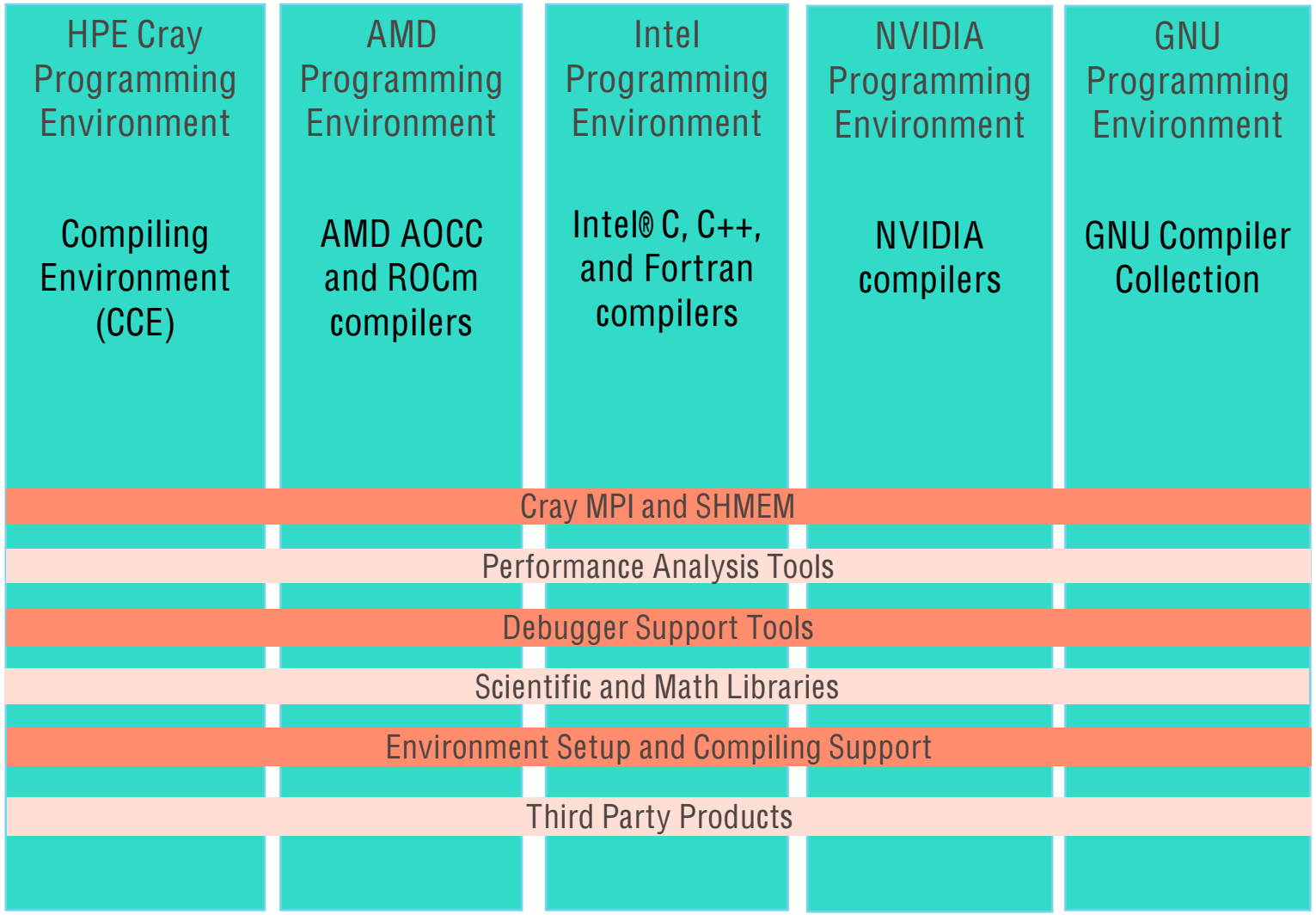
- Introduction
- Steps in moving an application to a GPU
- A simple example – himeno
- Optimizing himeno – with OpenMP on the node
- Optimizing himeno – with OpenMP Offoad (or OpenACC) for GPU
- Dealing with data movement
  - CPU <-> GPU
  - Halo Exchanges

Complete and fully supported software development Suite



Providing the user with compiler choice

- Use modules to select compiling environment
  - Automatically uses our math, scientific, and communication libraries with chosen compiler
  - Can use debug and profiling tools with chosen compiler

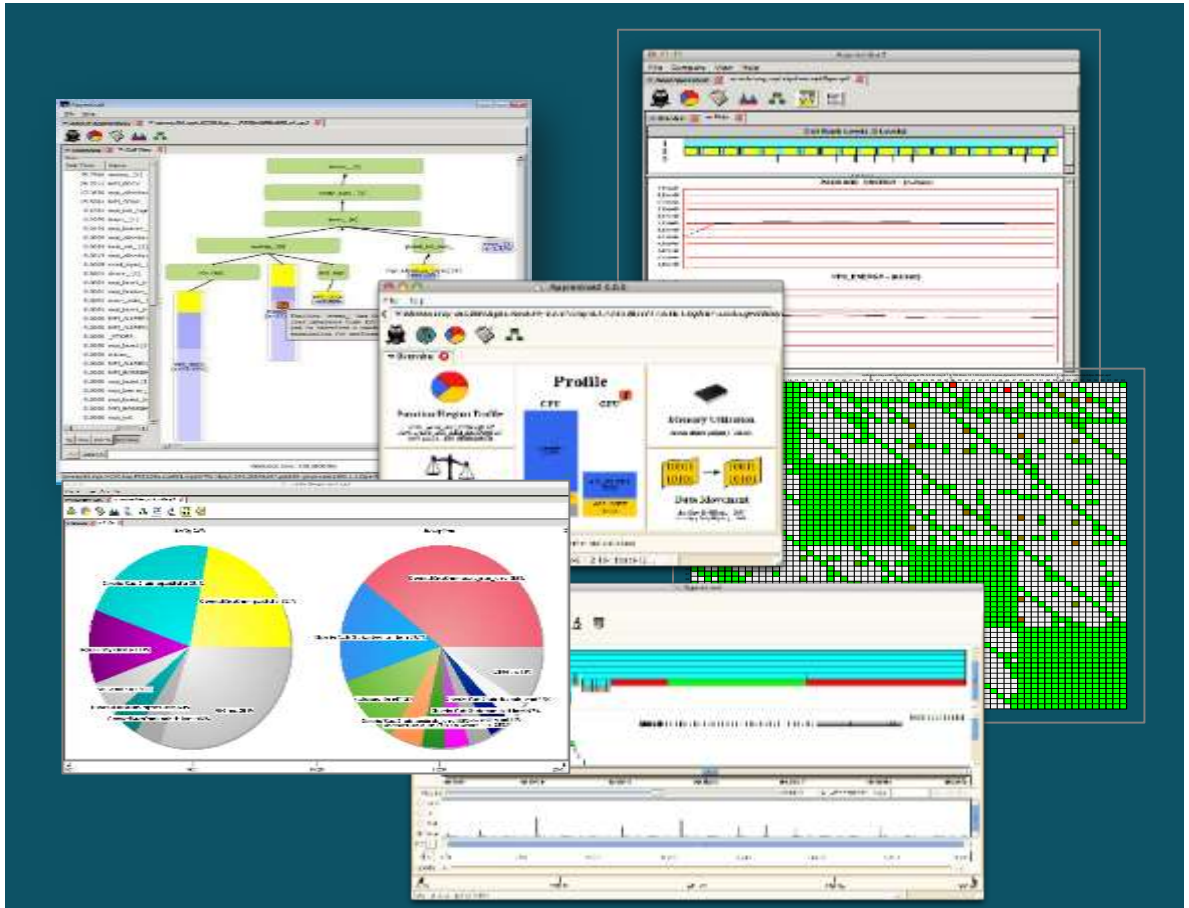


Performance Analysis Tools

Reduce time and effort associated with porting and tuning of applications on HPE systems

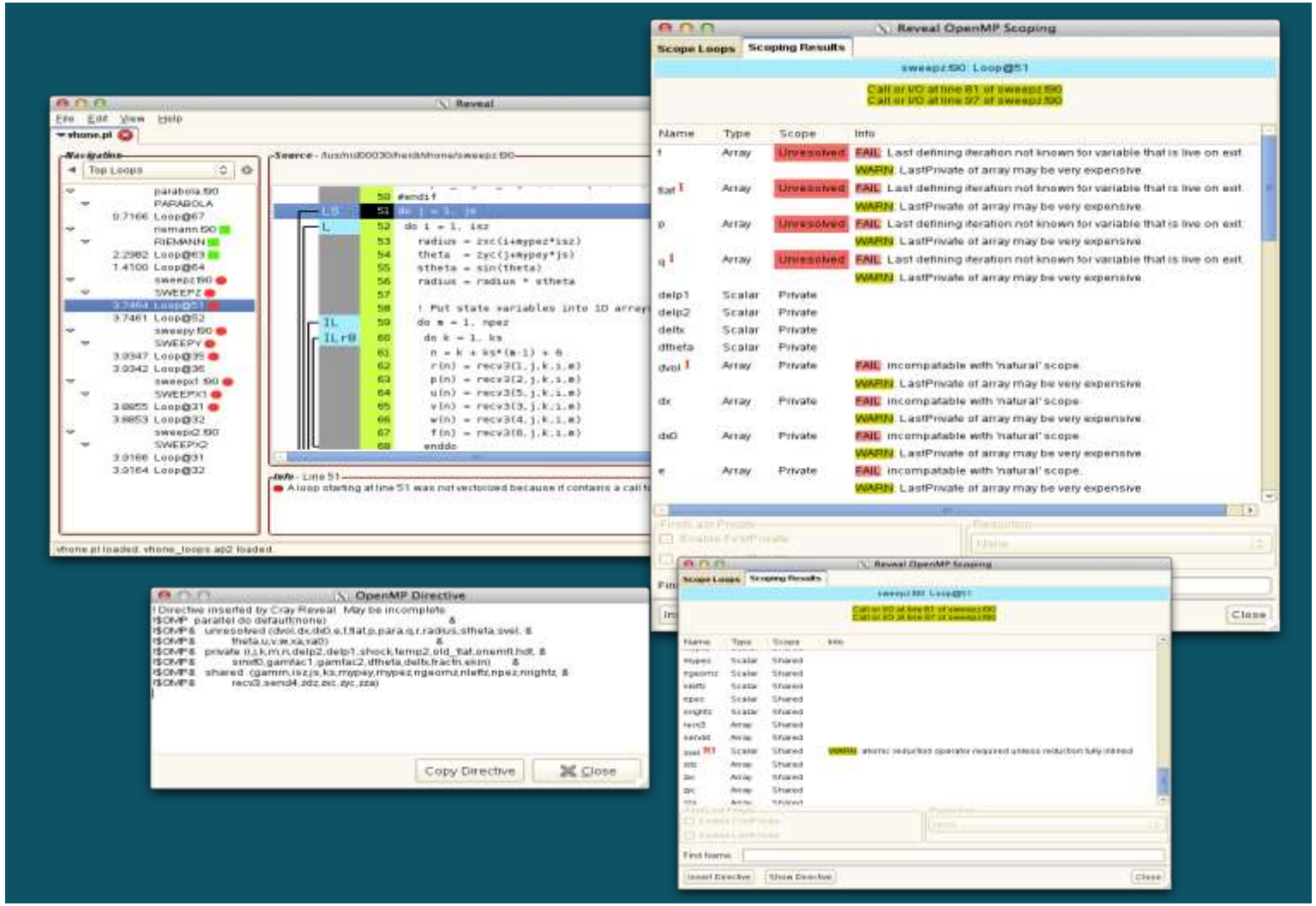
Highlights:

- Different tools to fit different developer needs—from quick visual analysis to variety of different experiments, integration with compilers and more…
- Target **scalability** issues in all areas of tool development—designed to improve performance on the largest of systems
- Provide **whole program performance analysis** across many nodes to identify critical performance bottlenecks in a program
- Help to uncover issues but also **suggestions to improve performance**
- Unique and valuable **load imbalance analysis**
- Target **ease of use** with simple and advanced user interfaces
- **Supports programs** written in Fortran, C or C++ with MPI, SHMEM, UPC, OpenMP or OpenACC, CUDA or HIP, and their combinations.



Our performance tools profiled production applications with over 256,000 ranks.

Code Parallelization Assistant



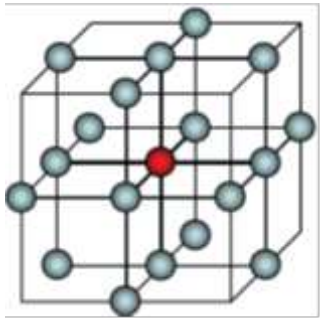
- Reduce effort associated with adding OpenMP to MPI programs
- Works in conjunction with our compiler and performance tools
- Easily navigate through source code to highlighted dependences or bottlenecks
- Identify work-intensive loops to parallelize, perform dependence analysis, scope variables and generate OpenMP directives
- Great first step when moving large, complex loops to GPUs

Directive-Based Programming Models

- Huge potential to provide cross-architecture portability (CPUs and GPUs)
- Standard specifications that all compiler vendors can implement
- Performance portability across vendors has been a recent challenge
- Has been critical for Fortran, especially for offloading
- OpenMP and OpenACC
  - Continued participation in language committees
  - Ongoing support for current and future specifications
  - Leverage common compiler and library codebase for OpenMP and OpenACC implementations
  - Significant opportunities for general performance improvements
  - Significant opportunities for improving construct-to-hardware mapping
    - Better cross-vendor consistency
    - Better use of descriptive features (e.g., “omp loop”)
    - Use of multidimensional grids, especially for “collapse” loops
- OpenMP offload and OpenACC support features to integrate with models such as CUDA and HIP
  - Optimize performance for a limited set of OpenMP/OpenACC constructs and APIs
  - Provide a portable model similar to existing kernel languages (e.g., CUDA or HIP)

Himeno benchmark

- 3D Poisson equation
  - 19-point stencil
  - Highly memory intensive, memory bound
- Fortran, C, MPI and OpenMP versions
  - Available from https://i.riken.jp/en/supercom/documents/himenobmt/
- Variously sized configurations



Himeno Benchmark kernel

- The stencil is applied to pressure array p
- Updated pressure values are saved to temporary array wrk2
- Control value wgosa is computed
- In the benchmark this kernel is iterated a fixed number of times (nn)

```
DO K=2,kmax-1
 DO J=2,jmax-1
  DO I=2,imax-1
   S0=a(I,J,K,1)*p(I+1,J, K )
     +a(I,J,K,2)*p(I, J+1,K ) &
     +a(I,J,K,3)*p(I, J, K+1) &
     +b(I,J,K,1)*(p(I+1,J+1,K )-p(I+1,J-1,K ) &
                 -p(I-1,J+1,K )+p(I-1,J-1,K )) &
     +b(I,J,K,2)*(p(I, J+1,K+1)-p(I, J-1,K+1) &
                 -p(I, J+1,K-1)+p(I, J-1,K-1)) &
     +b(I,J,K,3)*(p(I+1,J, K+1)-p(I-1,J, K+1) &
                 -p(I+1,J, K-1)+p(I-1,J, K-1)) &
     +c(I,J,K,1)*p(I-1,J, K ) &
     +c(I,J,K,2)*p(I, J-1,K ) &
     +c(I,J,K,3)*p(I, J, K-1) &
     + wrk1(I,J,K)

   SS=(S0*a(I,J,K,4)-p(I,J,K))*bnd(I,J,K)
   WGOSA=WGOSA+SS*SS
   wrk2(I,J,K)=p(I,J,K)+OMEGA *SS
  ENDDO
 ENDDO
ENDDO
```

Distributed version of Himeno

- The outer loop is performed a fixed number of times
- The Jacobi kernel is executed and new pressure array wrk2 and control value wgosa are computed
- The array is updated with the new pressure values
- The halo region values are exchanged between neighbor PEs
- Send and receive buffers are used
- The maximum control value is computed with an Allreduce operation across all the PEs

```
DO loop = 1, nn
   compute Jacobi kernel → wrk2,wgosa

   copy back wrk2 into p

   pack halo from p into send buffers

   exchange halos with neighbour PEs

   unpack halo into p from recv buffers

   Allreduce to sum wgosa across PEs
ENDDO
```

Using perftools-lite (or perftools-lite-loops or perftools-lite-hbm)

- Perftools-lite modules are the simplest way to run a performance experiment
- Before building an application:
  - `module load perftools-lite` or `perftools-lite-loops` or `perftools-lite-hbm`
  - Module perftools-base should already be loaded
- Build application
- Run application
- Statistics report is output to standard output at end of application execution
  - Also generates a directory of profile data which can be used to generate reports with different options

Let's start with a simple example

```
% cd F_mpi
% module load perftools-lite

% ftn -O3 -rm -hpl=himeno.pl -I.  -c himeno.f90 -o himeno.o
% ftn -O3 -rm -hpl=himeno.pl himeno.o -o himeno.exe
INFO: creating the PerfTools-instrumented executable 'himeno.exe' (lite-samples) ...OK

% cat job.slurm
#!/bin/bash
#SBATCH -t 10
#SBATCH -N 1
#SBATCH --ntasks-per-node=8
#SBATCH --exclusive

DIST="--cpu-bind=mask_cpu:0xfe,0xfe00,0xfe0000,0xfe000000,0xfe00000000,0xfe0000000000,0xfe000000000000,0xfe00000000000000"

export LD_LIBRARY_PATH=${CRAY_LD_LIBRARY_PATH}:${LD_LIBRARY_PATH}

time srun ${DIST} himeno.exe+orig > my_output_orig.${SLURM_JOBID} 2>&1
time srun ${DIST} himeno.exe > my_output.${SLURM_JOBID} 2>&1

% sbatch job.slurm

Files after run
himeno.exe+125664-8747781s  himeno.lst  my_output.3458017  my_output_orig.3458017 slurm-3458017.out
```

Produce annotated compiler listing

Compiler listing

directory with profile files

Himeno output

```
% cat my_output.*
CrayPat/X:  Version 23.03.0 Revision 46f710008  02/13/23 20:24:04
 Sequential version array size
  mimax= 1025  mjmax= 513  mkmax= 513
 Parallel version  array size
  mimax= 515  mjmax= 259  mkmax= 259
  imax= 513  jmax= 257  kmax= 257
  I-decomp=  2  J-decomp=  2  K-decomp=  2

  Start rehearsal measurement process.
  Measure the performance in 10  times.
   MFLOPS: 69617.461462995881   time(s): 1.2982310590000452,  4.340651794E-4
 Now, start the actual measurement process.
 The loop will be excuted in 50  times.
 This will take about one minute.
 Wait for a while.
  cpu  6.467868152000392 sec
  Loop executed for  50  times
  Gosa : 4.232053179E-4
  MFLOPS measured : 69868.11465231188
  Score based on Pentium III 600MHz : 843.4104
```

Perftools-lite profile

```
Table 1:  Profile by Function

  Samp% |  Samp | Imb. |  Imb. | Group
        |       | Samp | Samp% |  Function=[MAX10]
        |       |      |       |   PE=HIDE

 100.0% | 841.0 |   -- |    -- | Total
|-------------------------------------------------
|  80.8% | 679.6 |   -- |    -- | USER
||------------------------------------------------
||  75.8% | 637.5 |  3.5 |  0.6% | jacobi_
||   5.0% |  42.1 |  3.9 |  9.6% | himenobmtxp_
||================================================
|  17.0% | 142.9 |   -- |    -- | ETC
||------------------------------------------------
||  14.1% | 119.0 |  3.0 |  2.8% | __cray_memcpy_ROME
||   2.8% |  23.2 |  4.8 | 19.4% | __cray_memset_ROME
||================================================
|   2.2% |  18.5 |   -- |    -- | MPI
||------------------------------------------------
||   1.3% |  10.9 |  6.1 | 41.2% | MPI_WAITALL
|================================================
```

Exclusive time

Sampling is in

100th of a second

Imbalance

Only showing items that take up more than 1% of time – you can override with -T in pat_report

Perftools-lite profile

```
Table 1:  Profile by Function

  Samp% |  Samp | Imb. |   Imb. | Group
        |       | Samp |  Samp% |  Function
        |       |      |        |   PE=HIDE

 100.0% | 841.0 |   -- |     -- | Total
|-----------------------------------------------
|  80.8% | 679.6 |   -- |     -- | USER
||----------------------------------------------
||  75.8% | 637.5 |  3.5 |   0.6% | jacobi_
||   5.0% |  42.1 |  3.9 |   9.6% | himenobmtxp_
||==============================================
|  17.0% | 142.9 |   -- |     -- | ETC
||----------------------------------------------
||  14.1% | 119.0 |  3.0 |   2.8% | __cray_memcpy_ROME
||   2.8% |  23.2 |  4.8 |  19.4% | __cray_memset_ROME
||   0.0% |   0.2 |  0.8 |  85.7% | __cray_scopy_detect
||   0.0% |   0.1 |  0.9 | 100.0% | _init
||   0.0% |   0.1 |  0.9 | 100.0% | __cray_sset_ROME
||   0.0% |   0.1 |  0.9 | 100.0% | __cray_scopy_ROME
||==============================================
|   2.2% |  18.5 |   -- |     -- | MPI
||----------------------------------------------
||   1.3% |  10.9 |  6.1 |  41.2% | MPI_WAITALL
||   0.7% |   5.5 |  1.5 |  24.5% | MPI_ISEND
||   0.2% |   1.5 |  1.5 |  57.1% | MPI_IRECV
||   0.1% |   0.5 |  0.5 |  57.1% | MPI_BARRIER
||   0.0% |   0.1 |  0.9 | 100.0% | mpi_irecv_
|==============================================
```

pat_report -T himeno.exe+125664-8747781s > profile_T

Exclusive time

Sampling is in

100th of a second

Imbalance

Perftools-lite profile

```
Table 2:  Profile of maximum function times

  Samp% |  Samp | Imb. |   Imb. | Function
        |       | Samp |  Samp% |  PE=[max,min]
|-------------------------------------------------
| 100.0% | 641.0 |  3.5 |   0.6% | jacobi_
||------------------------------------------------
|| 100.0% | 641.0 |   -- |     -- | pe.3
||  98.4% | 631.0 |   -- |     -- | pe.6
||================================================
|  19.0% | 122.0 |  3.0 |   2.8% | __cray_memcpy_ROME
||------------------------------------------------
||  19.0% | 122.0 |   -- |     -- | pe.0
||  18.3% | 117.0 |   -- |     -- | pe.5
||================================================
|   7.2% |  46.0 |  3.9 |   9.6% | himenobmtxp_
||------------------------------------------------
||   7.2% |  46.0 |   -- |     -- | pe.3
||   5.9% |  38.0 |   -- |     -- | pe.6
||================================================
|   4.4% |  28.0 |  4.8 |  19.4% | __cray_memset_ROME
||------------------------------------------------
||   4.4% |  28.0 |   -- |     -- | pe.1
||   3.0% |  19.0 |   -- |     -- | pe.4
||================================================
…
```

pat_report -T himeno.exe+99198-8749895s > profile_T

Exclusive time

Sampling is in

100th of a second

Imbalance

Perftools-lite profile

```
================================================================================
  Total
--------------------------------------------------------------------------------
  Thread Time                                                  51.414680 secs
  UNHALTED_REFERENCE_CYCLES                          117,110,852,571
  CPU_CLK_THREAD_UNHALTED:THREAD_P                   181,647,161,792
  INST_RETIRED:ANY_P                                 109,619,182,320
  RESOURCE_STALLS:ANY                                145,947,632,023
  FP_ARITH:PACKED                                     38,585,536,000
  OFFCORE_RESPONSE_0:ANY_REQUEST:L3_MISS_LOCAL         4,480,911,135
  CPU CLK Boost                                                 1.55 X
  Resource stall cycles / Cycles                               80.3%
  FP Packed Instr / All Instr                                  35.2%
  Memory traffic GBytes                    5.578G/sec       286.78 GB
  Local Memory traffic GBytes              5.578G/sec       286.78 GB
  Memory Traffic / Nominal Peak                                 7.3%
  Retired Inst per Clock                                        0.60
================================================================================
```

Perftools-lite profile

```
Notes for table 5:

  This table show the average time and number of bytes written to each
    output file, taking the average over the number of ranks that
    wrote to the file.  It also shows the number of write operations,
    and average rates.
  For further explanation, see the "General table notes" below, or
    use:  pat_report -v -O write_stats ...

Table 5:  File Output Stats by Filename

      Avg |      Avg |  Write Rate | Number |    Avg | Bytes/ | File Name
    Write |    Write | MiBytes/sec |     of | Writes |   Call |  PE=HIDE
 Time per |  MiBytes |             | Writer |    per |        |
   Writer |      per |             |  Ranks | Writer |        |
     Rank |   Writer |             |        |   Rank |        |
          |     Rank |             |        |        |        |
|-----------------------------------------------------------------------
| 0.000014 | 0.000008 |    0.540886 |      8 |    1.0 |   8.00 | stderr
| 0.000004 | 0.000628 |  152.051781 |      1 |   19.0 |  34.63 | stdout
|=======================================================================
```

Perftools-lite profile

```
Table 3:  Profile by Group, Function, and Line

  Samp% |  Samp | Imb. |   Imb. | Group
        |       | Samp |  Samp% |  Function
        |       |      |        |   Source
        |       |      |        |    Line
        |       |      |        |     PE=HIDE

 100.0% | 841.0 |   -- |     -- | Total
|-----------------------------------------------------------
|  80.8% | 679.6 |   -- |     -- | USER
||----------------------------------------------------------
||  75.8% | 637.5 |   -- |     -- | jacobi_
3|        |       |      |        |  cug23/himeno_new/F_mpi/himeno.f90
||||--------------------------------------------------------
4|||   1.1% |   9.4 |  7.6 |  51.3% | line.222
4|||  46.8% | 393.4 | 13.6 |   3.8% | line.223
4|||  15.9% | 133.5 | 17.5 |  13.2% | line.229
4|||   0.6% |   5.4 |  5.6 |  58.4% | line.230
4|||  11.2% |  94.0 |  3.0 |   3.5% | line.231
4|||   0.2% |   1.8 |  2.2 |  64.3% | line.232
4|||   0.0% |   0.1 |  0.9 | 100.0% | line.239
||||========================================================
||   5.0% |  42.1 |  3.9 |   9.6% | himenobmtxp_
3|        |       |      |        |  cug23/himeno_new/F_mpi/himeno.f90
4|        |       |      |        |   line.65
||========================================================
```

Compiler loopmark information (file.lst, generated from –rm/-hlist=a options)

```
203.                    SUBROUTINE jacobi(nn,gosa)
204.                    !***************************************************************
205.
206.                    IMPLICIT REAL*4(a-h,o-z)
207.
208.                    INTEGER, INTENT(IN)                    :: nn
209.                    REAL, INTENT(OUT)                      :: gosa
210.
211.                    INTEGER*4 istat
212.
213.                    INCLUDE 'mpif.h'
214.                    INCLUDE 'param.h'
215.
216.  + 1---------------< DO loop=1,nn
217.    1                  gosa=0.0
218.    1                  wgosa=0.0
219.    1
220.  + 1 2-------------< DO k=2,kmax-1
221.  + 1 2 3-----------<   DO j=2,jmax-1
222.    1 2 3 Vr3-------<     DO i=2,imax-1
223.    1 2 3 Vr3               s0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k) +a(i,j,k,3)*p(i,j,k+1)  &
224.    1 2 3 Vr3                   +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k))  &
225.    1 2 3 Vr3                   +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1) -p(i,j+1,k-1)+p(i,j-1,k-1))  &
226.    1 2 3 Vr3                   +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1) -p(i+1,j,k-1)+p(i-1,j,k-1))  &
227.    1 2 3 Vr3                   +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)  &
228.    1 2 3 Vr3                   +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
229.    1 2 3 Vr3               ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
230.    1 2 3 Vr3               wgosa=wgosa+ss*ss
231.    1 2 3 Vr3               wrk2(i,j,k)=p(i,j,k)+omega *ss
232.    1 2 3 Vr3------->     END DO
233.    1 2 3----------->   END DO
234.    1 2------------> END DO
```

Vr3
Vectorised and unrolled by 3

Compiler loopmark **information…**

```
235.     1
236.   + 1 2-------------< DO k=2,kmax-1
237.   + 1 2 3-----------<   DO j=2,jmax-1
238.     1 2 3 A---------<     DO i=2,imax-1
239.     1 2 3 A                 p(i,j,k)=wrk2(i,j,k)
240.     1 2 3 A--------->     END DO
241.     1 2 3----------->   END DO
242.     1 2------------> END DO
243.     1
244.     1 I              CALL sendp(ndx,ndy,ndz)
245.     1
246.   + 1                CALL mpi_allreduce(wgosa, gosa,  &
247.     1                     1, mpi_real4,  &
248.     1                     mpi_sum, mpi_comm_world,  &
249.     1                     ierr)
250.     1
251.     1--------------> END DO
252.                      ! End of iteration
253.                      RETURN
254.                      END SUBROUTINE jacobi
```

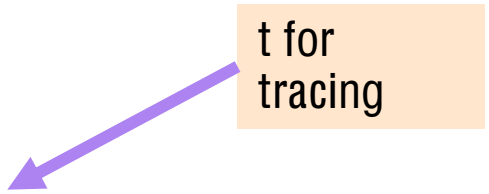
Loopmark Legend

Primary Loop Type		Modifiers	
A	Pattern matched	a	Atomic memory operation
		b	Blocked
C	Collapsed	c	Conditional and/or computed
D	Deleted		
E	Closed		
F	Flat – No calls	f	Fused
G	Accelerated	g	Partitioned
I	Inlined	i	Interchanged
M	Multithreaded	m	Partitioned
		n	Non-blocking remote transfer
		p	Partial
R	Rerolling	r	Unrolled
		s	Shortloop
V	Vectorized	w	Unwound

Let's Optimize this code

- Shared memory parallelism
- Then later move to the GPU
- To do these things we need to understand the loop limits (we want to go for loops that matter)

```
% module load perftools-lite-loops
% ftn -O3 -rm -hpl=himeno.pl -I.  -c himeno.f90 -o himeno.o
% ftn -O3 -rm -hpl=himeno.pl himeno.o -o himeno.exe
INFO: creating the PerfTools-instrumented executable 'himeno.exe' (lite-loops) ...OK
$ sbatch job.slurm
```



Files after run

```
himeno.exe+27714-8747979t  himeno.lst  my_output.3458104
my_output_orig.3458104  slurm-3458104.out
```

Perftools-lite-loops profile

pat_report -T –Oct himeno.exe+27714-8747979t

```
Table 1:  Calltree with Loop Inclusive Time

   Incl | Incl | Loop Exec |  Loop | Calltree
  Time% | Time |           | Trips |  PE=HIDE
        |      |           |   Avg |

 100.0% | 8.38 |        -- |    -- | Total
|-----------------------------------------------------------
| 100.0% | 8.38 |        -- |    -- | himenobmtxp_
||----------------------------------------------------------
||  92.4% | 7.74 |        -- |    -- | jacobi_
|||---------------------------------------------------------
3||  92.4% | 7.74 |         2 |  30.0 | jacobi_.LOOP.1.li.216
||||--------------------------------------------------------
4|||  74.5% | 6.25 |        60 | 255.0 | jacobi_.LOOP.2.li.220
|||||-------------------------------------------------------
5||||  74.5% | 6.25 |    15,300 | 255.0 | jacobi_.LOOP.3.li.221
||||||------------------------------------------------------
6|||||  74.0% | 6.20 | 3,901,500 | 511.0 | jacobi_.LOOP.4.li.222
|||||=======================================================
4|||  14.6% | 1.22 |        60 | 255.0 | jacobi_.LOOP.5.li.236
|||||-------------------------------------------------------
5||||  14.6% | 1.22 |    15,300 | 255.0 | jacobi_.LOOP.6.li.237
||||||------------------------------------------------------
6|||||   9.7% | 0.82 | 3,901,500 | 511.0 | jacobi_.LOOP.7.li.238
|||=======================================================
```

Inclusive time now since we are tracing and not sampling

Perftools-lite-**loops profile…**

pat_report -T –Oct himeno.exe+27714-8747979t

Table 1: Calltree with Loop Inclusive Time

Incl Time%	Incl Time	Loop Exec Trips Avg	Loop Trips	Calltree PE=HIDE
7.5%	0.63	--	--	initmt_
5.3%	0.44	1	259.0	initmt_.LOOP.1.li.156
5.3%	0.44	259	259.0	initmt_.LOOP.2.li.157
5.2%	0.44	67,081	515.0	initmt_.LOOP.3.li.158
2.3%	0.19	1	257.0	initmt_.LOOP.4.li.177
2.3%	0.19	257	257.0	initmt_.LOOP.5.li.178
2.2%	0.18	66,049	513.0	initmt_.LOOP.6.li.179
0.0%	0.00	--	--	initcomm_
0.0%	0.00	0	--	mpi_init_(sync)
0.0%	0.00	0	--	MPI_INIT
0.0%	0.00	0	--	_STOP3
0.0%	0.00	0	--	mpi_finalize_(sync)
0.0%	0.00	--	--	initmax_
0.0%	0.00	1	2.0	initmax_.LOOP.5.li.374
0.0%	0.00	1	2.0	initmax_.LOOP.1.li.344
0.0%	0.00	1	2.0	initmax_.LOOP.3.li.359
0.0%	0.00	1	2.0	initmax_.LOOP.2.li.351
0.0%	0.00	1	2.0	initmax_.LOOP.4.li.366
0.0%	0.00	1	2.0	initmax_.LOOP.6.li.381
0.0%	0.00	0	--	MPI_FINALIZE
0.0%	0.00	0	--	call_init.part.0

Crucial computation loops: compiler loopmark info

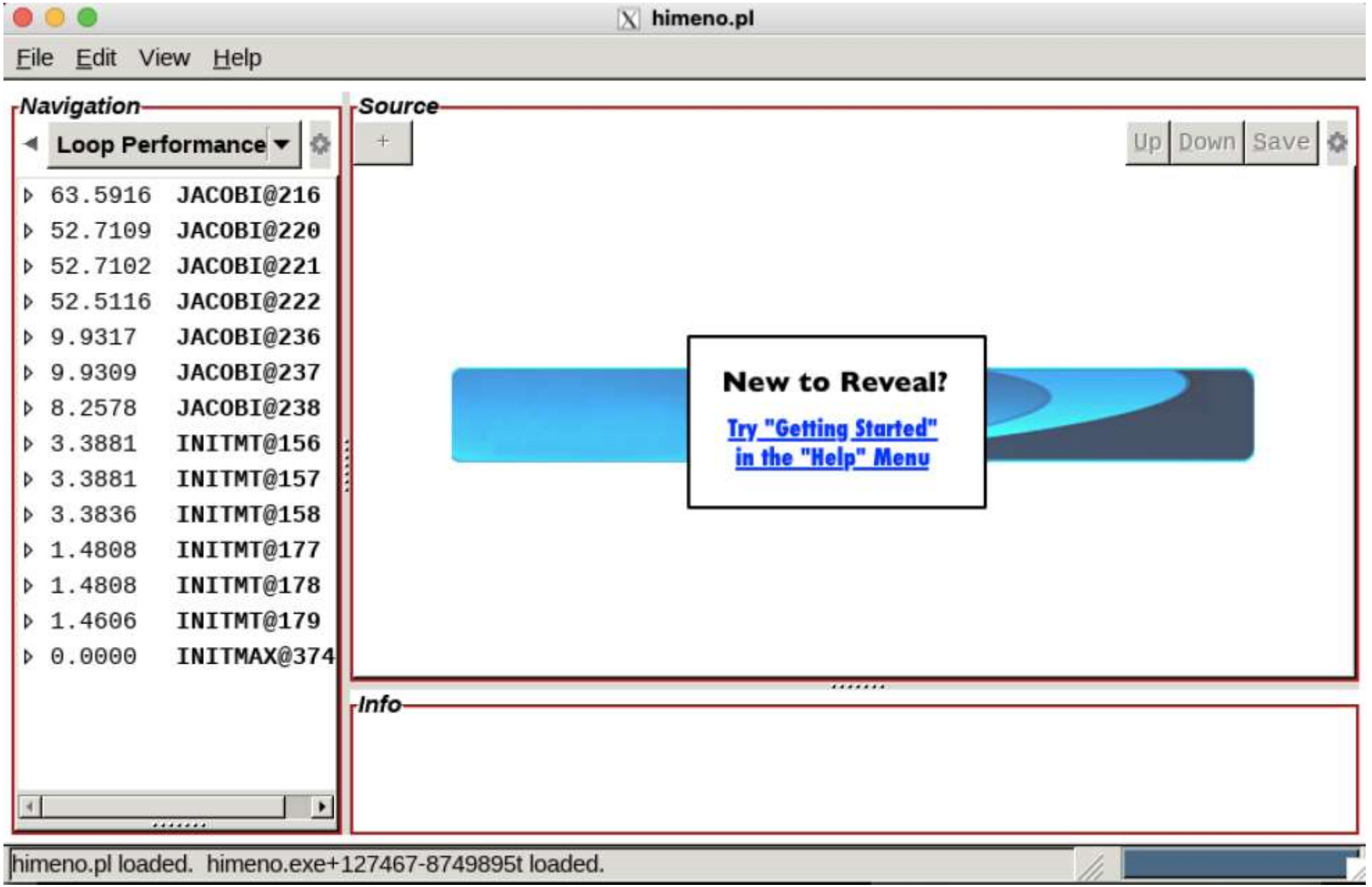
```
203.                  SUBROUTINE jacobi(nn,gosa)
204.                  !************************************************************
205.
206.                  IMPLICIT REAL*4(a-h,o-z)
207.
208.                  INTEGER, INTENT(IN)                    :: nn
209.                  REAL, INTENT(OUT)                      :: gosa
210.
211.                  INTEGER*4 istat
212.
213.                  INCLUDE 'mpif.h'
214.                  INCLUDE 'param.h'
215.
216.  + 1-----------< DO loop=1,nn
217.    1               gosa=0.0
218.    1               wgosa=0.0
219.    1
220.  + 1 2---------< DO k=2,kmax-1
221.  + 1 2 3-------<   DO j=2,jmax-1
222.    1 2 3 Vr3---<     DO i=2,imax-1
223.    1 2 3 Vr3           s0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k) +a(i,j,k,3)*p(i,j,k+1)  &
224.    1 2 3 Vr3               +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k))  &
225.    1 2 3 Vr3               +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1) -p(i,j+1,k-1)+p(i,j-1,k-1))  &
226.    1 2 3 Vr3               +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1) -p(i+1,j,k-1)+p(i-1,j,k-1))  &
227.    1 2 3 Vr3               +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)  &
228.    1 2 3 Vr3               +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
229.    1 2 3 Vr3           ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
230.    1 2 3 Vr3           wgosa=wgosa+ss*ss
231.    1 2 3 Vr3           wrk2(i,j,k)=p(i,j,k)+omega *ss
232.    1 2 3 Vr3--->     END DO
233.    1 2 3------->   END DO
234.    1 2--------> END DO
235.    1
236.  + 1 2---------< DO k=2,kmax-1
237.  + 1 2 3-------<   DO j=2,jmax-1
238.    1 2 3 A-----<     DO i=2,imax-1
239.    1 2 3 A             p(i,j,k)=wrk2(i,j,k)
240.    1 2 3 A----->     END DO
241.    1 2 3------->   END DO
242.    1 2--------> END DO
```

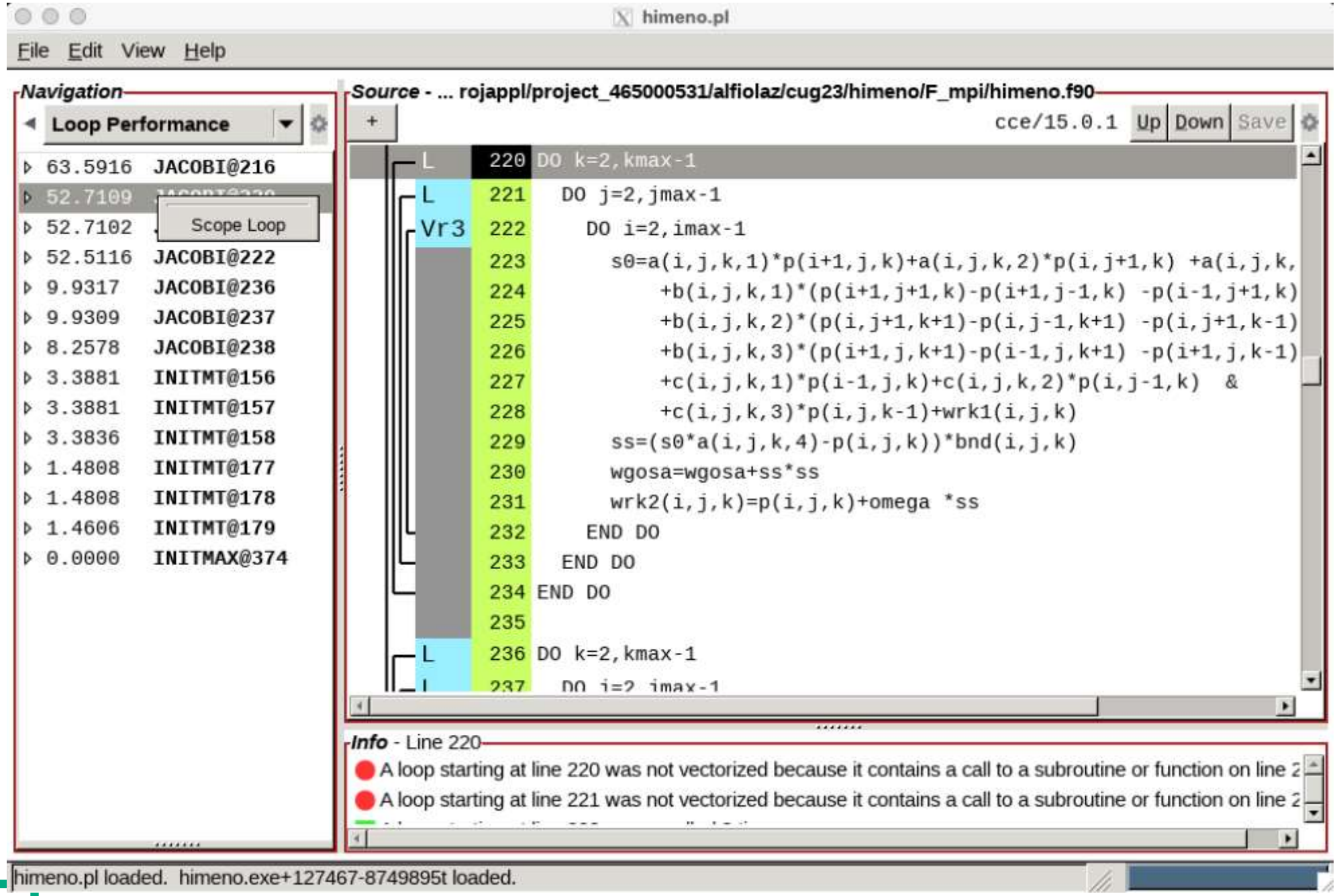
Let's optimize that code

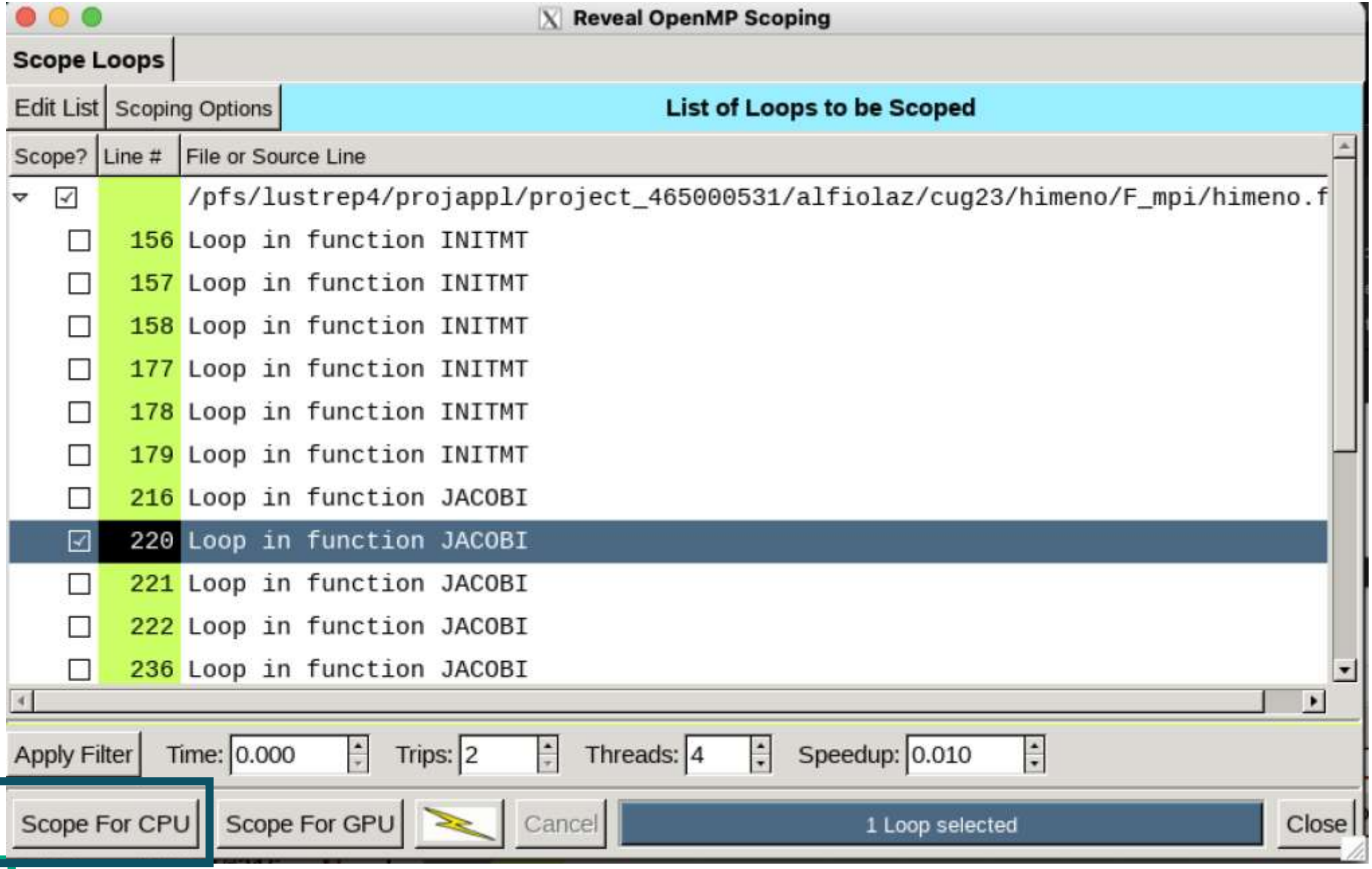
- Shared memory parallelism
- We have a tool called reveal to help parallelize code for host and GPU
- We need to build a program library to use reveal

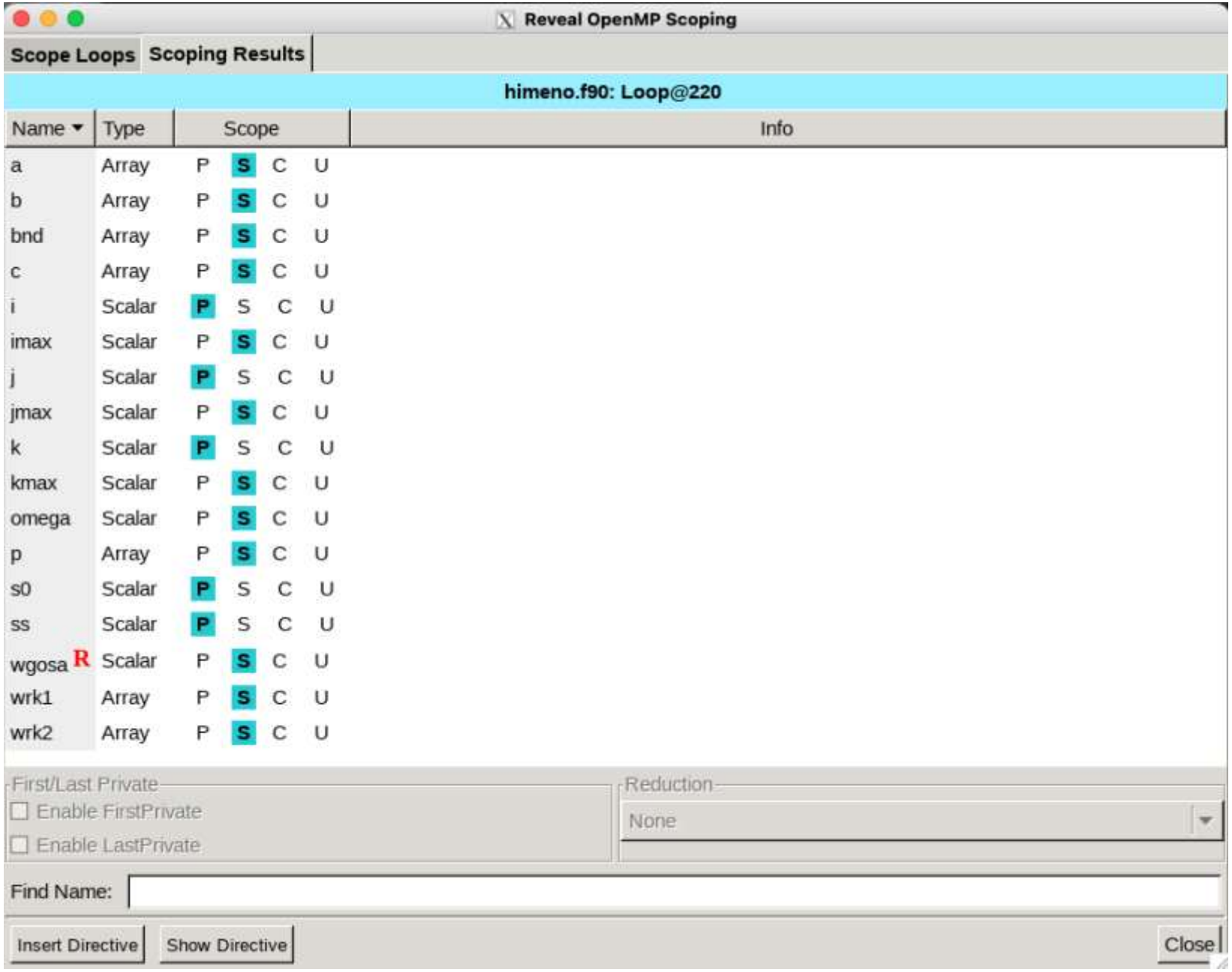
```
module unload perftools-lite-loops
reveal himeno.pl himeno.exe+27714-8747979t
```

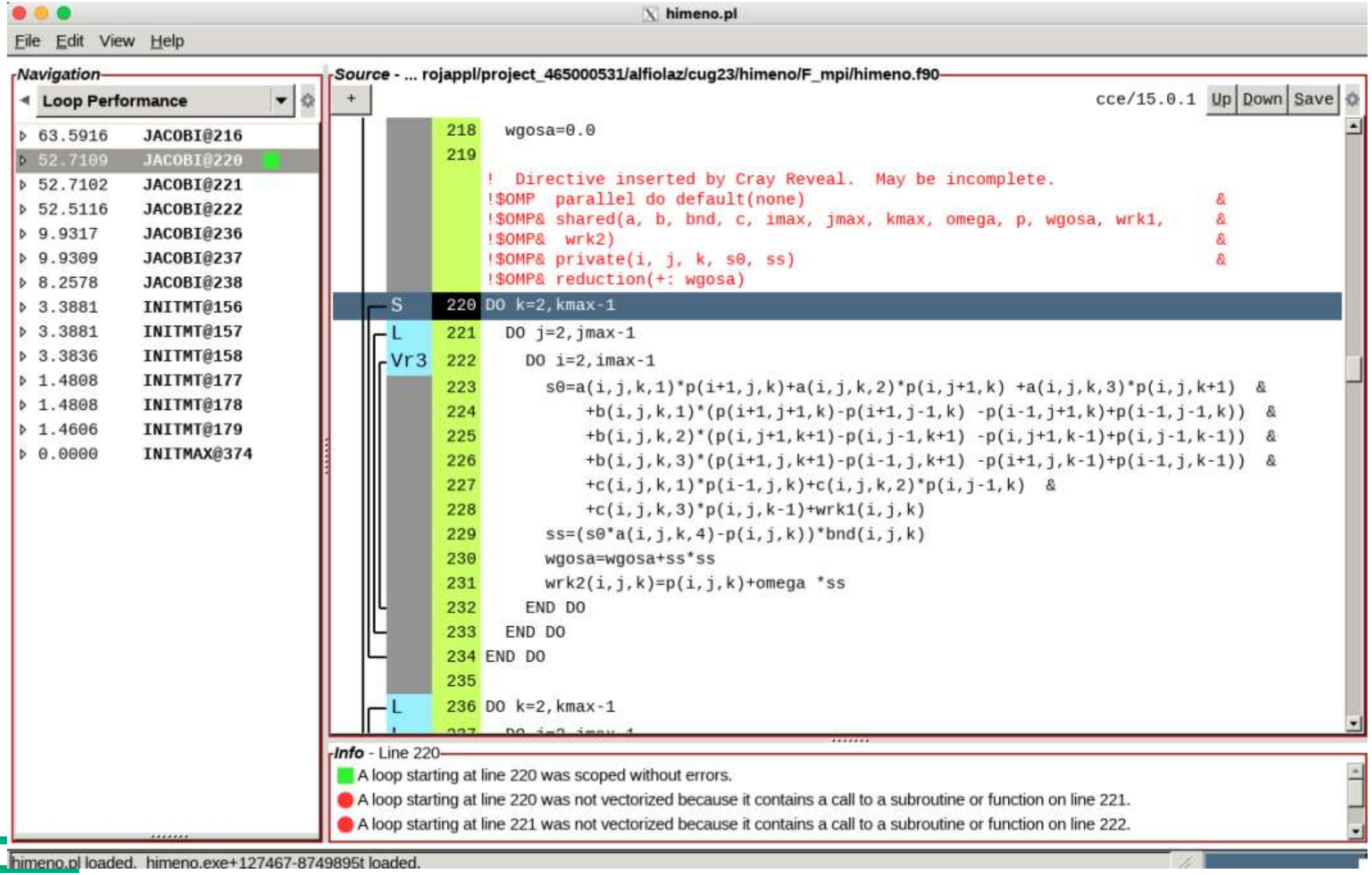
- The program library is needed so we can enable full inter-procedural analysis.









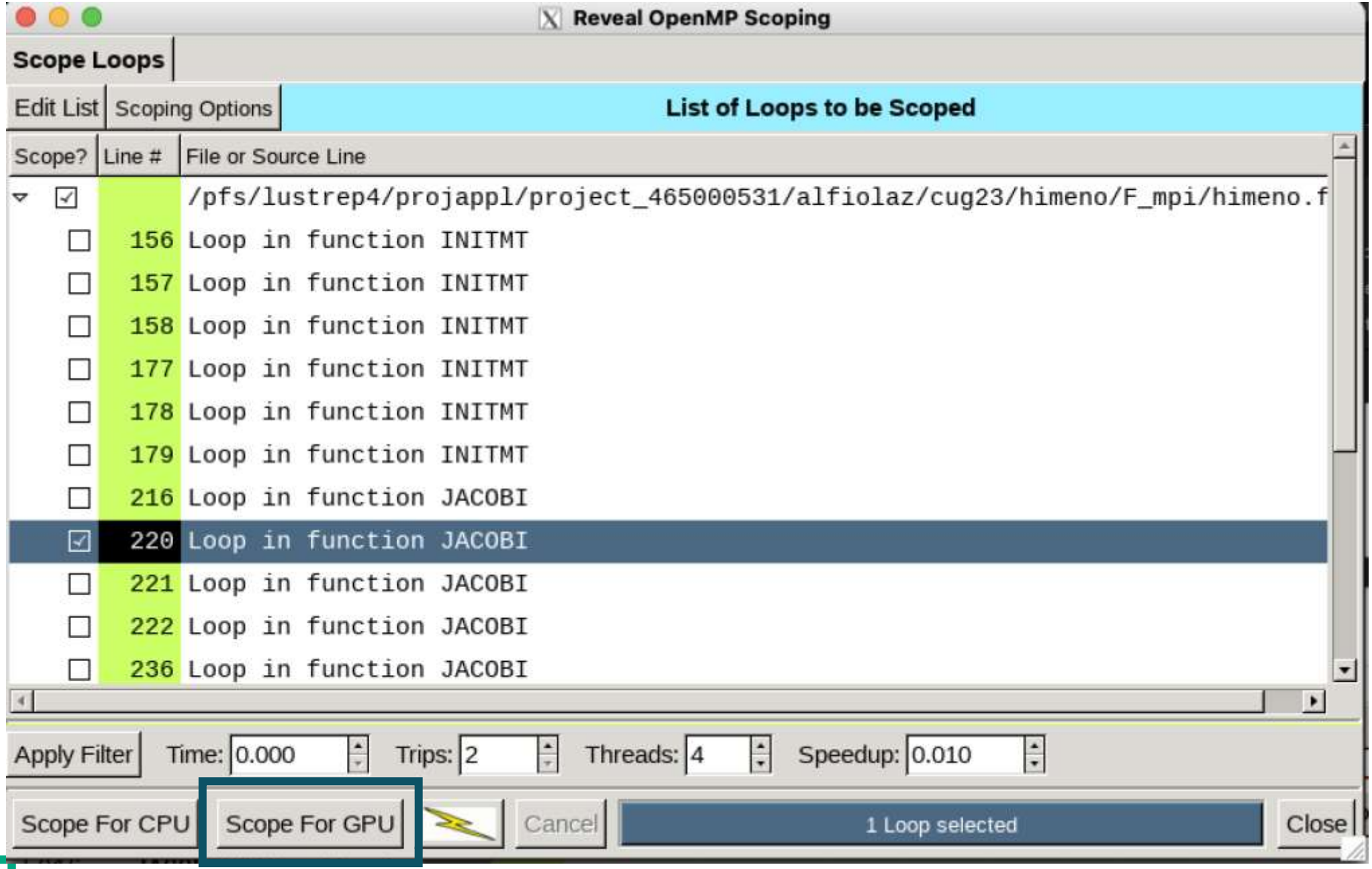

himeno.pl
File Edit View Help
Navigation
Loop Performance
63.5916 JACOBI@216
52.7109 JACOBI@220
52.7102 JACOBI@221
52.5116 JACOBI@222
9.9317 JACOBI@236
9.9309 JACOBI@237
8.2578 JACOBI@238
3.3881 INITMT@156
3.3881 INITMT@157
3.3836 INITMT@158
1.4808 INITMT@177
1.4808 INITMT@178
1.4606 INITMT@179
0.0000 INITMAX@374
Source - ... rojappl/project_465000531/alfiolaz/cug23/himeno/F_mpi/himeno.f90
cce/15.0.1 Up Down Save
218 wgosa=0.0
219
! Directive inserted by Cray Reveal. May be incomplete.
!$OMP parallel do default(none) &
!$OMP& shared(a, b, bnd, c, imax, jmax, kmax, omega, p, wgosa, wrk1, &
!$OMP& wrk2) &
!$OMP& private(i, j, k, s0, ss) &
!$OMP& reduction(+: wgosa)
S 220 DO k=2,kmax-1
L 221 DO j=2,jmax-1
Vr3 222 DO i=2,imax-1
223 s0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k) +a(i,j,k,3)*p(i,j,k+1) &
224 +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k)) &
225 +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1) -p(i,j+1,k-1)+p(i,j-1,k-1)) &
226 +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1) -p(i+1,j,k-1)+p(i-1,j,k-1)) &
227 +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k) &
228 +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
229 ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
230 wgosa=wgosa+ss*ss
231 wrk2(i,j,k)=p(i,j,k)+omega *ss
232 END DO
233 END DO
234 END DO
235
L 236 DO k=2,kmax-1
Info - Line 220
A loop starting at line 220 was scoped without errors.
A loop starting at line 220 was not vectorized because it contains a call to a subroutine or function on line 221.
A loop starting at line 221 was not vectorized because it contains a call to a subroutine or function on line 222.
himeno.pl loaded. himeno.exe+127467-8749895t loaded.

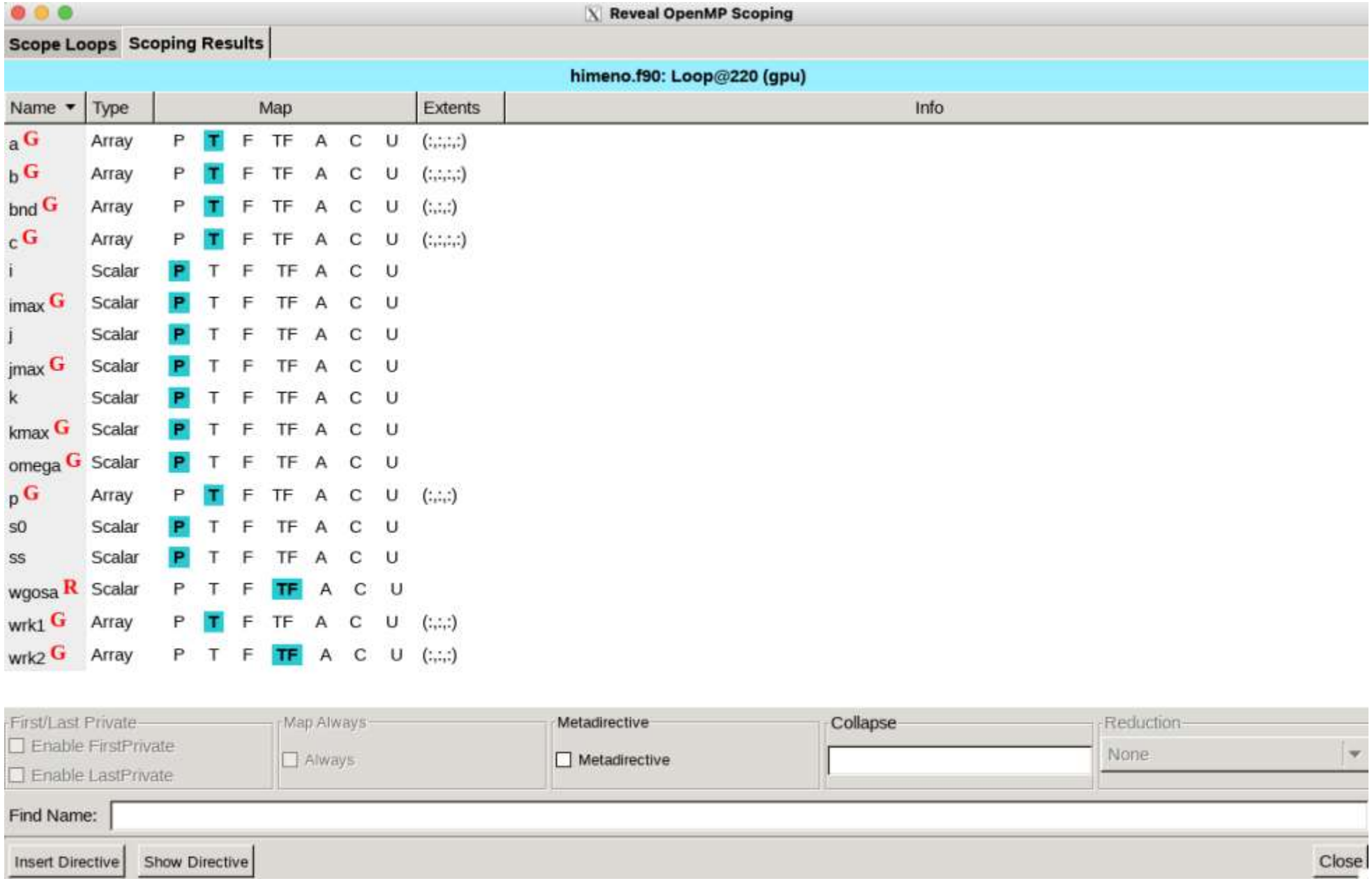
Threading performance improvement

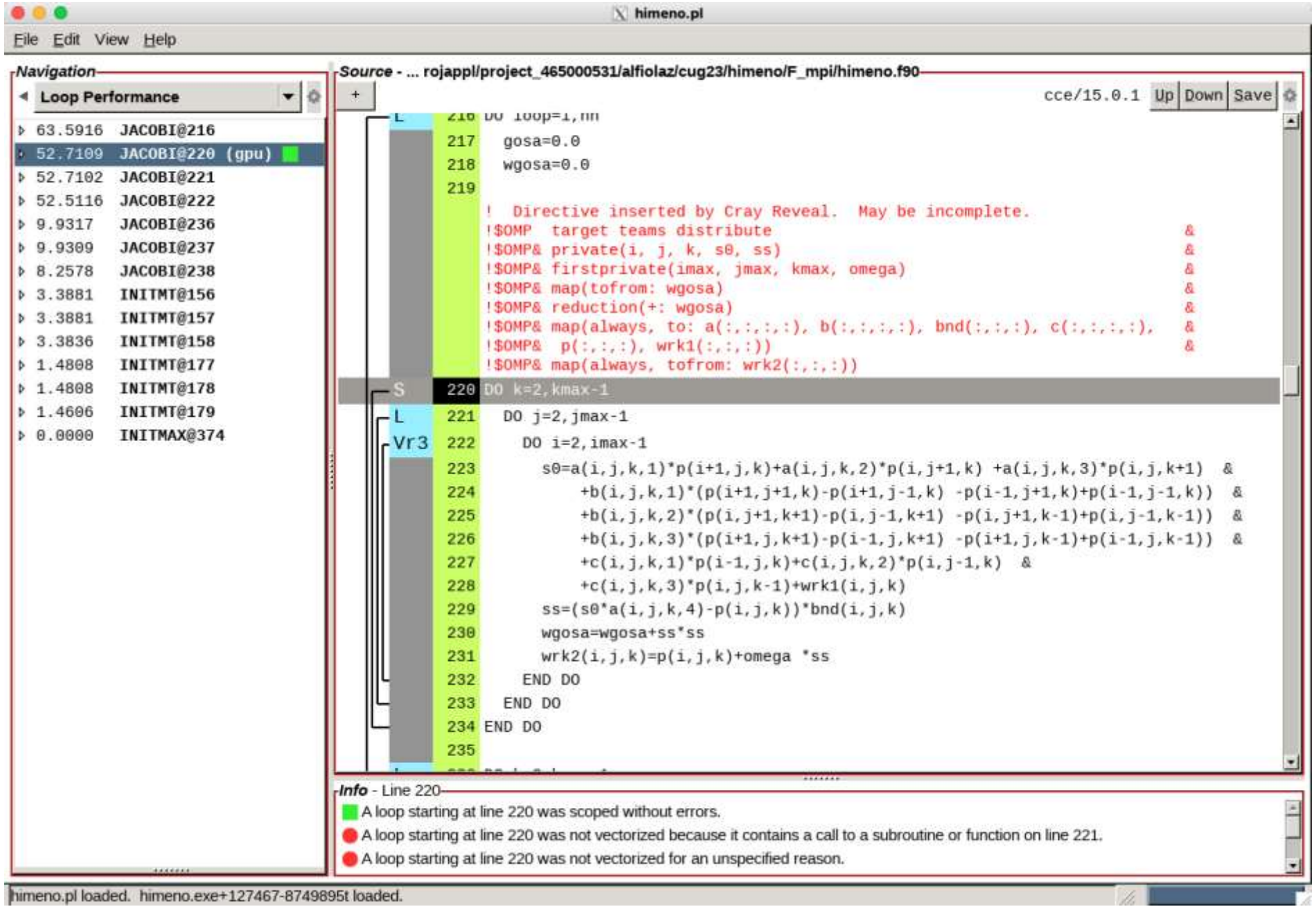
- We fix the number of iterations to be 50, so that we can compare time to solutions for different versions
- Remember that the benchmark is memory-bandwidth limited
- Still OpenMP gave a small benefit
  - 8% with 2 threads with respect to 1 thread on LUMI

Now let's put it on the GPU

- Go back to Reveal and generate GPU code


Reveal OpenMP Scoping
Scope Loops
Edit List
Scoping Options
List of Loops to be Scoped
Scope?
Line #
File or Source Line
/pfs/lustrep4/projappl/project_465000531/alfiolaz/cug23/himeno/F_mpi/himeno.f
156 Loop in function INITMT
157 Loop in function INITMT
158 Loop in function INITMT
177 Loop in function INITMT
178 Loop in function INITMT
179 Loop in function INITMT
216 Loop in function JACOBI
220 Loop in function JACOBI
221 Loop in function JACOBI
222 Loop in function JACOBI
236 Loop in function JACOBI
Apply Filter
Time: 0.000
Trips: 2
Threads: 4
Speedup: 0.010
Scope For CPU
Scope For GPU
Cancel
1 Loop selected
Close


Reveal OpenMP Scoping
Scope Loops
Scoping Results
himeno.f90: Loop@220 (gpu)
Name
Type
Map
Extents
Info
a G Array P T F TF A C U (:,:,:,:)
b G Array P T F TF A C U (:,:,:,:)
bnd G Array P T F TF A C U (:,:,:)
c G Array P T F TF A C U (:,:,:,:)
i Scalar P T F TF A C U
imax G Scalar P T F TF A C U
j Scalar P T F TF A C U
jmax G Scalar P T F TF A C U
k Scalar P T F TF A C U
kmax G Scalar P T F TF A C U
omega G Scalar P T F TF A C U
p G Array P T F TF A C U (:,:,:)
s0 Scalar P T F TF A C U
ss Scalar P T F TF A C U
wgosa R Scalar P T F TF A C U
wrk1 G Array P T F TF A C U (:,:,:)
wrk2 G Array P T F TF A C U (:,:,:)
First/Last Private
Enable FirstPrivate
Enable LastPrivate
Map Always
Always
Metadirective
Metadirective
Collapse
Reduction
None
Find Name:
Insert Directive
Show Directive
Close


himeno.pl
File Edit View Help
Navigation
Loop Performance
63.5916 JACOBI@216
52.7109 JACOBI@220 (gpu)
52.7102 JACOBI@221
52.5116 JACOBI@222
9.9317 JACOBI@236
9.9309 JACOBI@237
8.2578 JACOBI@238
3.3881 INITMT@156
3.3881 INITMT@157
3.3836 INITMT@158
1.4808 INITMT@177
1.4808 INITMT@178
1.4606 INITMT@179
0.0000 INITMAX@374
Source - ... rojappl/project_465000531/alfiolaz/cug23/himeno/F_mpi/himeno.f90
cce/15.0.1 Up Down Save
217 gosa=0.0
218 wgosa=0.0
219
! Directive inserted by Cray Reveal. May be incomplete.
!$OMP target teams distribute &
!$OMP& private(i, j, k, s0, ss) &
!$OMP& firstprivate(imax, jmax, kmax, omega) &
!$OMP& map(tofrom: wgosa) &
!$OMP& reduction(+: wgosa) &
!$OMP& map(always, to: a(:,:,:,:), b(:,:,:,:), bnd(:,:,:), c(:,:,:,:), &
!$OMP& p(:,:,:), wrk1(:,:,:)) &
!$OMP& map(always, tofrom: wrk2(:,:,:))
S 220 DO k=2,kmax-1
L 221 DO j=2,jmax-1
Vr3 222 DO i=2,imax-1
223 s0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k) +a(i,j,k,3)*p(i,j,k+1) &
224 +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k)) &
225 +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1) -p(i,j+1,k-1)+p(i,j-1,k-1)) &
226 +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1) -p(i+1,j,k-1)+p(i-1,j,k-1)) &
227 +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k) &
228 +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
229 ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
230 wgosa=wgosa+ss*ss
231 wrk2(i,j,k)=p(i,j,k)+omega *ss
232 END DO
233 END DO
234 END DO
235
Info - Line 220
A loop starting at line 220 was scoped without errors.
A loop starting at line 220 was not vectorized because it contains a call to a subroutine or function on line 221.
A loop starting at line 220 was not vectorized for an unspecified reason.
himeno.pl loaded. himeno.exe+127467-8749895t loaded.

Initial performance

- Performance obtained on LUMI (a single node):

CPU only (8 ranks x 2 threads):	75 GFLOPs
Initial GPU porting (8 ranks, 8 GPUs):	28 GFLOPs

- Why is GPU execution performing worse?
  - Check data movement!

- Perftools-lite-gpu specifically target GPU profiling

```
% module load craype-accel-amd-gfx90a rocm
% module load perftools-lite-gpu
% pat_report –T <directory generated by execution>
```

Perftools-lite-gpu profile

```
Table 1:  Profile by Function Group and Function

  Time% |      Time |     Imb. |   Imb. |    Calls | Group
        |           |     Time |  Time% |          |  Function
        |           |          |        |          |   PE=HIDE
        |           |          |        |          |    Thread=HIDE

 100.0% | 22.400068 |       -- |     -- | 22,546.6 | Total
|-------------------------------------------------------------------------
|  86.5% | 19.372358 |       -- |     -- | 20,617.0 | OACC
||------------------------------------------------------------------------
||  77.1% | 17.266184 | 0.836747 |   5.3% |  3,904.0 | cray_acc_transfer_list
||   9.4% |  2.099139 | 0.030002 |   1.6% | 16,473.0 | acc_get_device_num
||   0.0% |  0.005400 | 0.001280 |  21.9% |     60.0 | jacobi_.ACC_COPY@li.221
||   0.0% |  0.000964 | 0.000331 |  29.2% |     60.0 | jacobi_.ACC_COPY@li.243
||   0.0% |  0.000388 | 0.000073 |  18.0% |     60.0 | jacobi_.ACC_ASYNC_KERNEL@li.221
||   0.0% |  0.000284 | 0.000041 |  14.4% |     60.0 | jacobi_.ACC_SYNC_WAIT@li.243
||========================================================================
|   7.4% |  1.665387 |       -- |     -- |     63.0 | USER
||------------------------------------------------------------------------
||   4.5% |  1.005416 | 0.149915 |  14.8% |      2.0 | jacobi_
||   2.9% |  0.659721 | 0.007622 |   1.3% |      1.0 | himenobmtxp_
||   0.0% |  0.000249 | 0.000058 |  21.5% |     60.0 | jacobi_.ACC_REGION@li.221
||========================================================================
|   3.7% |  0.819459 |       -- |     -- |    983.0 | MPI
||------------------------------------------------------------------------
…
```

Perftools-lite-gpu profile

Table 9: Time and Bytes Transferred for Accelerator Regions

Time%	Time	Acc Time%	Acc Time	Acc Copy In (MiBytes)	Acc Copy Out (MiBytes)	Events	Calltree Accelerator ID PE=HIDE Thread=HIDE
100.0%	22.400068	100.0%	17.23	221,399	15,814	3,725	Total
100.0%	22.400035	100.0%	17.23	221,399	15,814	3,725	himenobmtxp_
95.8%	21.467652	100.0%	17.23	221,399	15,814	3,725	jacobi_
78.3%	17.530224	100.0%	17.23	221,399	15,814	3,724	jacobi_.ACC_REGION@li.221
72.2%	16.180262	92.2%	15.89	221,399	--	2,484	jacobi_.ACC_COPY@li.221
72.2%	16.174863	--	--	110,700	--	2,424	cray_acc_transfer_list acc.0
0.0%	0.005400	92.2%	15.89	110,700	--	60	jacobi_.ACC_COPY@li.221(exclusive) acc.0
4.9%	1.092285	6.3%	1.09	--	15,814	660	jacobi_.ACC_COPY@li.243
4.9%	1.091321	--	--	--	7,907	600	cray_acc_transfer_list acc.0
0.0%	0.000964	6.3%	1.09	--	7,907	60	jacobi_.ACC_COPY@li.243(exclusive) acc.0

…

Compiler loopmark information

Copy data h2d per each iteration!

```
216.  + 1----------------< DO loop=1,nn
217.    1                   gosa=0.0
218.    1                   wgosa=0.0
219.    1
220.    1                 !  Directive inserted by Cray Reveal.  May be incomplete.
221.  + 1 MG-------------< !$OMP  target teams distribute                                  &
222.    1 MG              !$OMP& private(i, j, k, s0, ss)                                 &
223.    1 MG              !$OMP& firstprivate(imax, jmax, kmax, omega)                    &
224.    1 MG              !$OMP& map(tofrom: wgosa)                                       &
225.    1 MG              !$OMP& reduction(+: wgosa)                                      &
226.    1 MG              !$OMP& map(always, to: a(:,:,:,:), b(:,:,:,:), bnd(:,:,:), c(:,:,:,:),   &
227.    1 MG              !$OMP&  p(:,:,:), wrk1(:,:,:))                                  &
228.    1 MG              !$OMP& map(always, tofrom: wrk2(:,:,:))
229.    1 MG g-----------< DO k=2,kmax-1
230.  + 1 MG g 4---------<   DO j=2,jmax-1
231.    1 MG g 4 gr3-----<     DO i=2,imax-1
232.    1 MG g 4 gr3             s0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k) +a(i,j,k,3)*p(i,j,k+1)  &
233.    1 MG g 4 gr3                 +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k))  &
234.    1 MG g 4 gr3                 +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1) -p(i,j+1,k-1)+p(i,j-1,k-1))  &
235.    1 MG g 4 gr3                 +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1) -p(i+1,j,k-1)+p(i-1,j,k-1))  &
236.    1 MG g 4 gr3                 +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)  &
237.    1 MG g 4 gr3                 +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
238.    1 MG g 4 gr3             ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
239.    1 MG g 4 gr3             wgosa=wgosa+ss*ss
240.    1 MG g 4 gr3             wrk2(i,j,k)=p(i,j,k)+omega *ss
241.    1 MG g 4 gr3----->     END DO
242.    1 MG g 4--------->   END DO
243.    1 MG g---------->> END DO
```

Complications with Himeno

- All Arrays are in COMMON blocks – these are globally available, so we need a way to keep them on the GPU all the time.
- Reveal is putting data transfer inside the iteration loop
  - However, it does recognize that on P and WRK2 need to be updated
- Biggest issue is transferring the entire P array when only boundaries need to be exchanged
- Following slides outline some optimizations for himeno
  - The plan is to put all the arrays and operations on those arrays on the accelerator.
  - Whenever the host needs to do something with the accelerator data, we must update the host from the accelerator

Allocate the data on the GPU once: data environment

```
my= my0-1
mz= mz0-1

!$OMP TARGET DATA MAP (alloc:a,b,bnd,c,p,wrk1,wrk2)

!C Initializing communicator
CALL initcomm

!C Initializaing computational index
CALL initmax(mx,my,mz,it)

…

gosa= 0.0
cpu= 0.0
CALL mpi_barrier(mpi_comm_world,ierr)
cpu0= mpi_wtime()
! Jacobi iteration
CALL jacobi(nn,gosa)
cpu1= mpi_wtime() - cpu0

!$OMP END TARGET DATA
```

Initialize arrays on the accelerator

```
156.  + MG-----------------< !$OMP  target teams distribute           &
157.    MG                  !$OMP& collapse(3)                       &
158.    MG                  !$OMP& private(i, j, k)
159.    MG C---------------< DO k=1,mkmax
160.    MG C C-------------<   DO j=1,mjmax
161.    MG C C gCr2--------<     DO i=1,mimax
162.    MG C C gCr2                a(i,j,k,1)=0.0
163.    MG C C gCr2                a(i,j,k,2)=0.0
164.    MG C C gCr2                a(i,j,k,3)=0.0
165.    MG C C gCr2                a(i,j,k,4)=0.0
166.    MG C C gCr2                b(i,j,k,1)=0.0
167.    MG C C gCr2                b(i,j,k,2)=0.0
168.    MG C C gCr2                b(i,j,k,3)=0.0
169.    MG C C gCr2                c(i,j,k,1)=0.0
170.    MG C C gCr2                c(i,j,k,2)=0.0
171.    MG C C gCr2                c(i,j,k,3)=0.0
172.    MG C C gCr2 A-----<>       p(i,j,k)=0.0
173.    MG C C gCr2 A-----<>       wrk1(i,j,k)=0.0
174.    MG C C gCr2 A-----<>       wrk2(i,j,k)=0.0
175.    MG C C gCr2 A-----<>       bnd(i,j,k)=0.0
176.    MG C C gCr2-------->     END DO
177.    MG C C------------->   END DO
178.    MG C-------------->> END DO
```

Initialize arrays on the accelerator

```
180.  + MG-----------------< !$OMP  target teams distribute                          &
181.    MG                  !$OMP& collapse(3)                                       &
182.    MG                  !$OMP& private(i, j, k)
183.    MG C---------------< DO k=1,kmax
184.    MG C C-------------<   DO j=1,jmax
185.  + MG C C gCr8--------<     DO i=1,imax
186.    MG C C gCr8               a(i,j,k,1)=1.0
187.    MG C C gCr8               a(i,j,k,2)=1.0
188.    MG C C gCr8               a(i,j,k,3)=1.0
189.    MG C C gCr8               a(i,j,k,4)=1.0/6.0
190.    MG C C gCr8               b(i,j,k,1)=0.0
191.    MG C C gCr8               b(i,j,k,2)=0.0
192.    MG C C gCr8               b(i,j,k,3)=0.0
193.    MG C C gCr8               c(i,j,k,1)=1.0
194.    MG C C gCr8               c(i,j,k,2)=1.0
195.    MG C C gCr8               c(i,j,k,3)=1.0
196.    MG C C gCr8               p(i,j,k)=FLOAT((k-1+it)*(k-1+it)) /FLOAT((mz-1)*(mz-1))
197.    MG C C gCr8               wrk1(i,j,k)=0.0
198.    MG C C gCr8               wrk2(i,j,k)=0.0
199.    MG C C gCr8               bnd(i,j,k)=1.0
200.    MG C C gCr8-------->     END DO
201.    MG C C------------->   END DO
202.    MG C-------------->> END DO
```

Use these arrays in the main iterations

```
!$OMP  target teams distribute                                                  &
!$OMP& private(i, j, k, s0, ss)                                                 &
!$OMP& map(tofrom: wgosa) MAP(present,to:a,b,c,wrk1,wrk2,bnd)                   &
!$OMP& reduction(+: wgosa)
DO k=2,kmax-1
  DO j=2,jmax-1
    DO i=2,imax-1
      s0=a(i,j,k,1)*p(i+1,j,k)+a(i,j,k,2)*p(i,j+1,k) +a(i,j,k,3)*p(i,j,k+1)  &
          +b(i,j,k,1)*(p(i+1,j+1,k)-p(i+1,j-1,k) -p(i-1,j+1,k)+p(i-1,j-1,k))  &
          +b(i,j,k,2)*(p(i,j+1,k+1)-p(i,j-1,k+1) -p(i,j+1,k-1)+p(i,j-1,k-1))  &
          +b(i,j,k,3)*(p(i+1,j,k+1)-p(i-1,j,k+1) -p(i+1,j,k-1)+p(i-1,j,k-1))  &
          +c(i,j,k,1)*p(i-1,j,k)+c(i,j,k,2)*p(i,j-1,k)  &
          +c(i,j,k,3)*p(i,j,k-1)+wrk1(i,j,k)
      ss=(s0*a(i,j,k,4)-p(i,j,k))*bnd(i,j,k)
      wgosa=wgosa+ss*ss
      wrk2(i,j,k)=p(i,j,k)+omega *ss
    END DO
  END DO
END DO
```

Saying that these arrays are already mapped to the GPU

Swap arrays and update halos

```
!$OMP  target teams distribute  &
!$OMP& private(i, j, k)
DO k=2,kmax-1
  DO j=2,jmax-1
    DO i=2,imax-1
      p(i,j,k)=wrk2(i,j,k)
    END DO
  END DO
END DO

!$OMP TARGET UPDATE from(p)
CALL sendp(ndx,ndy,ndz)

CALL mpi_allreduce(wgosa, gosa,  &
    1, mpi_real4,  &
    mpi_sum, mpi_comm_world,  &
    ierr)

END DO
! End of iteration
```

Need to get p back to host for MPI

Performance with data regions

- Performance obtained on LUMI (a single node):

CPU only (8 ranks x 2 threads):	75 GFLOPs
Initial GPU porting (8 ranks, 8 GPUs):	28 GFLOPs
Data transfer optimized GPU porting (8 ranks, 8 GPUs):	368 GFLOPs

Let's deal with the Halo exchanges

- There are some options

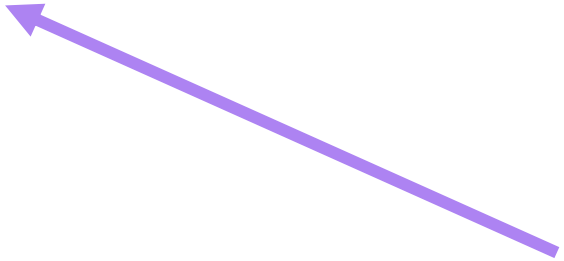
- We want to avoid copying the P array to do halo swaps
  - We can copy the halos to contiguous buffers on the GPU and only move those between GPU and CPU

- Next, we can consider directly communicating the halo data from GPU to GPU
  - This is supported in Cray MPICH if we pass GPU pointers to MPI

- Let's start with the buffer copying approach…

Halo updates

```
!$OMP TARGET DATA MAP (alloc:xr_s,xr_n,xs_s,xs_n)
      call mpi_irecv(XR_S(1),
     >               mjmax*mkmax,
     >               MPI_real,
     >               npx(1),
     >               2,
     >               mpi_comm_world,
     >               ireq(3),
     >               ierr)
c
      call mpi_irecv(XR_N(1),
     >               mjmax*mkmax,
     >               MPI_real,
     >               npx(2),
     >               1,
     >               mpi_comm_world,
     >               ireq(2),
     >               ierr)
c
!$OMP  target teams distribute map(from:xs_s,xs_n) private(icnt)
      DO K = 1,mkmax
       DO J = 1,mjmax
        ICNT = (K-1)*mjmax+J
        XS_S(ICNT)=P(2,J,K)
        XS_N(ICNT)=P(mimax-1,J,K)
       END DO
      END DO
!$OMP TARGET UPDATE from(xs_s,xs_n)
      call mpi_isend(XS_S(1),
     >               mjmax*mkmax,
     >               MPI_real,
     >               npx(1),
     >               1,
```



Buffers on GPU

Halo updates

```
c
      call mpi_waitall(4,
     >                 ireq,
     >                 ist,
     >                 ierr)
c
!$OMP TARGET UPDATE to(xr_s,xr_n)
!$OMP  target teams distribute map(from:xr_s,xr_n) private(icnt)
      DO K = 1,mkmax
       DO J = 1,mjmax
        ICNT = (K-1)*mjmax+J
        if(npx(1).ne.-1)P(mimax,J,K)= XR_S(ICNT)
        if(npx(2).ne.-1)P(1,J,K)= XR_N(ICNT)
       END DO
      END DO
!$OMP END TARGET DATA


      return
      end
```

Perftools profile

Table 1: Profile by Function Group and Function

Time%	Time	Imb. Time	Imb. Time%	Calls	Group Function PE=HIDE Thread=HIDE
100.0%	0.967625	--	--	31,655.0	Total
65.9%	0.637231	--	--	20,137.0	OACC
18.1%	0.174761	0.001591	1.2%	1.0	initmt_.ACC_COPY@li.152
17.6%	0.170772	0.000987	0.8%	500.0	jacobi_.ACC_SYNC_WAIT@li.240
4.8%	0.046532	0.000311	0.9%	500.0	jacobi_.ACC_SYNC_WAIT@li.250
3.3%	0.032382	0.000266	1.1%	14.0	jacobi_.ACC_COPY@li.87
2.7%	0.026012	0.000636	3.2%	503.0	sendp1_.ACC_COPY@li.679
2.7%	0.025881	0.000367	1.9%	503.0	sendp2_.ACC_COPY@li.623
2.7%	0.025762	0.000490	2.5%	503.0	sendp1_.ACC_COPY@li.703
2.6%	0.024681	0.000240	1.3%	503.0	sendp2_.ACC_COPY@li.598
1.1%	0.010461	0.000029	0.4%	1.0	jacobi_.ACC_COPY@li.212
1.0%	0.010089	0.000175	2.3%	500.0	jacobi_.ACC_COPY@li.240
0.9%	0.008922	0.000093	1.4%	500.0	jacobi_.ACC_COPY@li.217
0.8%	0.007708	0.000420	6.9%	503.0	sendp1_.ACC_SYNC_WAIT@li.678
0.8%	0.007384	0.000228	4.0%	503.0	sendp1_.ACC_COPY@li.704
0.8%	0.007291	0.000140	2.5%	503.0	sendp2_.ACC_COPY@li.624
0.5%	0.005304	0.000173	4.2%	503.0	sendp1_.ACC_ASYNC_KERNEL@li.704
0.5%	0.005155	0.000106	2.7%	503.0	sendp2_.ACC_ASYNC_KERNEL@li.624
0.5%	0.004888	0.000037	1.0%	500.0	jacobi_.ACC_ASYNC_KERNEL@li.217
0.5%	0.004387	0.000250	7.2%	503.0	sendp1_.ACC_ASYNC_KERNEL@li.671
0.4%	0.004315	0.000104	3.1%	500.0	jacobi_.ACC_ASYNC_KERNEL@li.242
0.4%	0.004123	0.000047	1.5%	503.0	sendp1_.ACC_SYNC_WAIT@li.711
0.4%	0.003967	0.000163	5.2%	503.0	sendp2_.ACC_ASYNC_KERNEL@li.590
0.4%	0.003436	0.000138	5.1%	503.0	sendp2_.ACC_SYNC_WAIT@li.597

Enabling GPU to GPU communication with MPI

- We can tell MPI to use GPU data
- We set MPICH_GPU_SUPPORT_ENABLED=1 for Cray MPICH
- We use the OpenMP use_device_ptr to specify we are using the variable on the GPU

- We can do this in the sendp<n> routines:

```
!$OMP TARGET DATA MAP(alloc:zr_e,zr_w,zs_e,zs_w)
!$OMP TARGET DATA use_device_ptr(zr_e,zr_w,zs_e,zs_w)
      call mpi_irecv(ZR_E(1),
     >               imax*jmax,
     >               MPI_real,
```

Replace the “target update” with the GPU device pointer, so that MPI will run the GPU-aware MPI

Enabling GPU to GPU communication with MPI

- It turns out that Cray MPICH can accept GPU pointers when communicating MPI datatypes
- So we could have just made our MPI calls and passed in the device pointer as follows

```
!$omp target data use_device_ptr(p)
! Call MPI calls for halo exchanges
CALL sendp(ndx,ndy,ndz)
!$omp end target data
```

Replace the “target update” with the GPU device pointer, so that MPI will run the GPU-aware MPI

- This does work with HPE Cray MPICH but MPI is not doing the packing on the GPU.

Performance with data regions and G2G MPI on p

- Performance obtained on LUMI (a single node):

CPU only (8 ranks x 2 threads):	75 GFLOPs
Initial GPU porting (8 ranks, 8 GPUs):	28 GFLOPs
Data transfer optimized GPU porting (8 ranks, 8 GPUs):	368 GFLOPs
Data transfer optimized GPU porting with G2G (8 ranks, 8 GPUs):	374 GFLOPs

CRAY_ACC_DEBUG Environment Variable

Three levels of verbosity

- 1 High Level overview of kernels executed and data transferred
- 2 Breaks down data transfer by each variable
- 3 The whole Kitchen sink

export CRAY_ACC_DEBUG=1

```
ACC: Transfer 29 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:21
ACC: Transfer 15 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:114
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:141
ACC: Transfer 29 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:21
ACC: Transfer 15 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:114
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:141
ACC: Execute kernel extrapi_$ck_L141_7 async(auto) from src/fluxi.f:141
ACC: Execute kernel extrapi_$ck_L141_7 async(auto) from src/fluxi.f:141
ACC: Wait async(auto) from src/fluxi.f:172
ACC: Transfer 1 items (to acc 0 bytes, to host 0 bytes) from src/fluxi.f:172
ACC: Execute kernel extrapi_$ck_L173_9 async(auto) from src/fluxi.f:173
ACC: Wait async(auto) from src/fluxi.f:172
```

export CRAY_ACC_DEBUG=2

```
ACC: Start transfer 29 items from src/fluxi.f:21
ACC:         present 'dq' (255923712 bytes)
ACC:         present 'ds' (95971392 bytes)
ACC:         present 'ds1' (95971392 bytes)
ACC:         present 'dtv' (31990464 bytes)
ACC:         present 'hf' (127961856 bytes)
ACC:         present 'pav' (31990464 bytes)
ACC:         present 'q' (511847424 bytes)
ACC:         present 'qav' (255923712 bytes)
ACC:         present 'six' (31990464 bytes)
ACC:         present 'siy' (31990464 bytes)
ACC:         present 'siz' (31990464 bytes)
ACC:         present 't' (31990464 bytes)
```

export CRAY_ACC_DEBUG=3

```
ACC: Start transfer 29 items from src/fluxi.f:21
ACC:   flags: RETURN_ACC_TIME
ACC:
ACC:   Trans 1
ACC:       Simple transfer of 'dq' (144 bytes)
ACC:            host ptr 78b538
ACC:            acc  ptr 0
ACC:            flags: DOPE_VECTOR DV_ONLY_DATA ALLOCATE COPY_HOST_TO_ACC ACQ_PRESENT
REG_PRESENT
ACC:            Transferring dope vector
ACC:              dim:1 lowbound:-2 extent:198 stride_mult:1
ACC:              dim:2 lowbound:-2 extent:198 stride_mult:198
ACC:              dim:3 lowbound:-2 extent:102 stride_mult:39204
ACC:              dim:4 lowbound:1 extent:8 stride_mult:3998808
ACC:              DV size=255923712 (scale:8, elem_size:8)
ACC:              total mem size=255923712 (dv:0 obj:255923712)
ACC:            host region 4dac8780 to 5ced9d80 found in present table index 0 (ref count 2)
ACC:            memory found in present table (2aaaf2000000, base 2aaaf2000000)
ACC:            new acc ptr 2aaaf2000000
```

Output from the initial GPU version (Jacobi iterations)

```
0: ACC: Version 5.0 of HIP already initialized, runtime version 50221153
0: ACC: Get Device 0
0: ACC: Set Thread Context
0: ACC: Start transfer 10 items from himeno.f90:221
0: ACC:       allocate, copy to acc 'a(:,:,:,:)' (552747440 bytes)
0: ACC:       allocate, copy to acc 'b(:,:,:,:)' (414560580 bytes)
0: ACC:       allocate, copy to acc 'bnd(:,:,:)' (138186860 bytes)
0: ACC:       allocate, copy to acc 'c(:,:,:,:)' (414560580 bytes)
0: ACC:       allocate, copy to acc 'p(:,:,:)' (138186860 bytes)
0: ACC:       allocate, copy to acc 'wgosa' (4 bytes)
0: ACC:       allocate, copy to acc 'wrk1(:,:,:)' (138186860 bytes)
0: ACC:       allocate, copy to acc 'wrk2(:,:,:)' (138186860 bytes)
0: ACC:       allocate reusable, copy to acc <internal> (4 bytes)
0: ACC:       allocate reusable <internal> (1020 bytes)
0: ACC: End transfer (to acc 1934616048 bytes, to host 0 bytes)
0: ACC: Execute kernel jacobi_$ck_L221_1 blocks:255 threads:256 async(auto) from himeno.f90:221
0: ACC: Wait async(auto) from himeno.f90:243
0: ACC: Start transfer 10 items from himeno.f90:243
0: ACC:       free 'a(:,:,:,:)' (552747440 bytes)
0: ACC:       free 'b(:,:,:,:)' (414560580 bytes)
0: ACC:       free 'bnd(:,:,:)' (138186860 bytes)
0: ACC:       free 'c(:,:,:,:)' (414560580 bytes)
0: ACC:       free 'p(:,:,:)' (138186860 bytes)
0: ACC:       copy to host, free 'wgosa' (4 bytes)
0: ACC:       free 'wrk1(:,:,:)' (138186860 bytes)
0: ACC:       copy to host, free 'wrk2(:,:,:)' (138186860 bytes)
0: ACC:       done reusable <internal> (4 bytes)
0: ACC:       done reusable <internal> (1020 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 138186864 bytes)
```

Repeated per each Jacobi iteration

We are moving a lot of data

Output from the GPU version with data movement optimized (initialization)

```
0: ACC: Version 5.0 of HIP already initialized, runtime version 50221153
0: ACC: Get Device 0
0: ACC: Set Thread Context
0: ACC: Start transfer 7 items from himeno.f90:58
0: ACC:      allocate 'a(:,:,:,:)' (552747440 bytes)
0: ACC:      allocate 'b(:,:,:,:)' (414560580 bytes)
0: ACC:      allocate 'bnd(:,:,:)' (138186860 bytes)
0: ACC:      allocate 'c(:,:,:,:)' (414560580 bytes)
0: ACC:      allocate 'p(:,:,:)' (138186860 bytes)
0: ACC:      allocate 'wrk1(:,:,:)' (138186860 bytes)
0: ACC:      allocate 'wrk2(:,:,:)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
0: ACC: Start transfer 7 items from himeno.f90:67
0: ACC:      present 'a(:,:,:,:)' (552747440 bytes)
0: ACC:      present 'b(:,:,:,:)' (414560580 bytes)
0: ACC:      present 'bnd(:,:,:)' (138186860 bytes)
0: ACC:      present 'c(:,:,:,:)' (414560580 bytes)
0: ACC:      present 'p(:,:,:)' (138186860 bytes)
0: ACC:      present 'wrk1(:,:,:)' (138186860 bytes)
0: ACC:      present 'wrk2(:,:,:)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
0: ACC: Execute kernel himenobmtxp_$ck_L67_1 blocks:134949 threads:256 async(auto) from himeno.f90:67
0: ACC: Wait async(auto) from himeno.f90:67
```

Output from GPU version with G2G (halo swaps)

```
0: ACC: Start transfer 1 items from himeno.f90:261
0: ACC:       present 'p(:,:,:)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
0: ACC: Start transfer 1 items from himeno.f90:264
0: ACC:       release present 'p(:,:,:)' (138186860 bytes)
0: ACC: End transfer (to acc 0 bytes, to host 0 bytes)
```

```
261 !$omp target data use_device_ptr(p)
262 ! Call MPI calls for halo exchanges
263 CALL sendp(ndx,ndy,ndz)
264 !$omp end target data
```

So, what have we learned?

- Perftools is excellent for identifying issues in existing applications for improving threading, vectorization and scalar optimization
- Reveal can help with the difficult job of scoping variables in potential parallelizable loops
  - More difficult if not impossible with C++
- Moving to the GPU is difficult; however, it can be done in steps that are more manageable
  - Perftools identifies the bottlenecks in the GPU application very quickly
  - Target the time-consuming parts of the application
  - Be concerned with data movement (keep data on GPU)
  - Only move data when needed
- GPU direct is best way to do the message passing – in this case it only matters at scale