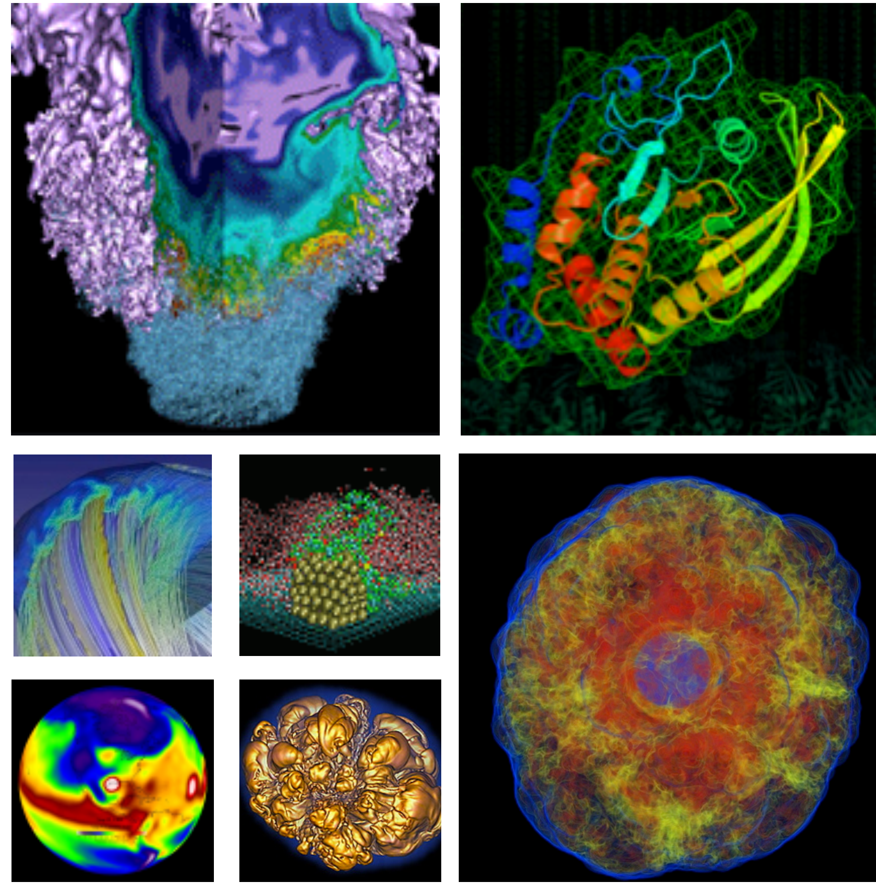
More Profiling Tools at NERSC



Woo-Sun Yang
NERSC User Services Group

NUG Training
February 25, 2015

Overview

- **To provide a quick start for profiling tools (other than VTune) on Edison**
  – CrayPat
    - Reveal (not a profiling tool)
  – CrayPat-lite
  – IPM (not covered)
  – MAP
  – HPCToolkit

Performance analysis tools

- **Measure code performance to identify performance bottlenecks and improve them**
- **Two types of measurement**
  - Sampling
    - Sample where the program is executing (i.e., 'program counter') at regular time intervals (or certain events)
    - Low overhead
    - Useful in most cases for identifying performance hotspots
  - Tracing
    - Focus on the selected functions to see detailed info on their usage
    - User specifies a list of the functions to be traced
      - CrayPat's APA (Automatic Program Analysis) suggests which functions to trace
    - Larger overhead, especially with functions which are called frequently
- **Some tools available at NERSC**
  - CrayPat: for sampling or tracing
  - IPM: for sampling
  - MAP: for sampling
  - HPCToolkit: for sampling

Workflow with CrayPat (Cray Performance Measurement and Analysis Tools)

1. **'module load perftools' before starting to build your code**
2. **Build your code; *.o must be kept as well as *.a, if any**
3. **Instrument your executable using 'pat_build'**
   – **pat_build [options] a.out** # to create an instrumented binary, # a.out+pat
4. **Execute your instrumented program**
   – **aprun … ./a.out+pat** # in a batch job
   – Collected data saved in **a.out+pat+#####-####*e*.xf**
   (*e*: **s** for sampling and **t** for tracing)
5. **Analyze the resulting data**
   – **pat_report a.out+pat+#####-####*e*.xf**

CrayPat run types

- **Instrumentation types**
  - Sampling
  - Tracing: Specify a list of the functions to be traced
    - User functions: using pat_build's -T,-t, -u (-u for all; can increase run time significantly)
    - Preset trace groups for popular functions: using pat_build's -g
      - mpi, heap, io, omp, blas, lapack, …
- **Sampling run traces MPI functions, some system functions, etc. by default**
- **Sampling run automatically generates a .apa file that contains pat_build flags to trace suggested functions and function groups for a tracing run**

Sampling with CrayPat

```
$ module rm darshan
$ module load perftools
$ ftn -c -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -o bgw.x
$ pat_build -f bgw.x
$ cat runit
...
 aprun -n 1 ./bgw.x+pat
$ qsub runit
2446538.edique02
$ pat_report bgw.x+pat+48499-6113s.xf > my.rpt
$ more my.rpt
$ app2 bgw.x+pat+48499-6113s.ap2
$ rm bgw.x+pat+48499-6113s.xf
```

Unload darshan as it will interfere with perftools

Build an instr. binary; -f to overwrite if there is one already

Use the instr. binary

ASCII text report captured in my.rpt

See the report

Visualization of the results using a GUI tool, app2

Not needed as you now have a .ap2 file;
*.ap2 is self-contained and portable while .xf is not;
text report can be generated from .ap2, too

```
$ ls -l *.apa
-rw-------   1 wyang wyang    1799 Feb 24 14:22 bgw.x+pat+48499-6113s.apa
```

This text file contains pat_build options to generate an instrumented executable for a tracing run with suggested list of trace functions and function groups; see the next slide

Tracing with CrayPat (one way - using Automatic Program Analysis)

```
$ module rm darshan
$ module load perftools
$ ftn -c -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -o bgw.x
```

Unload darshan as it will interfere with perftools

```
$ vi bgw.x+pat+48499-6113s.apa
$ pat_build -f -O bgw.x+pat+48499-6113s.apa
```

Edit suggested trace functions/groups, if you want

Build a new instr. binary for tracing, guided by the sampling results

```
$ cat runit
...
#export PAT_RT_SUMMARY=0
 aprun -n 1 ./bgw.x+apa
$ qsub runit
2447437.edique02
```

For more detailed data; data size can become huge

Use the new instr. binary for tracing

```
$ pat_report bgw.x+apa+32565-3822t.xf > myt.rpt
$ more myt.rpt
$ app2 bgw.x+apa+32565-3822t.ap2
$ rm *.xf
```

ASCII text report in myt.rpt

If you want…

Not needed as you now have .ap2 files

CrayPat results in the text report

```
% more my.rpt
...
Table 2:  Profile by Group, Function, and Line

  Samp% |    Samp | Imb. |  Imb. |Group
        |         | Samp | Samp% | Function
        |         |      |       |  Source
        |         |      |       |   Line

 100.0% | 5714.0 |   -- |    -- |Total
|------------------------------------------------------------
|  98.7% | 5639.0 |   -- |    -- |USER
||-----------------------------------------------------------
||  98.7% | 5639.0 |   -- |    -- |MAIN__
3|        |        |      |       | scratchdirs/wyang/BGW/bgw.f90
||||---------------------------------------------------------
4|||    2.6% |  146.0 |   -- |    -- |line.71
4|||    2.5% |  143.0 |   -- |    -- |line.73
4|||   17.4% |  996.0 |   -- |    -- |line.177
4|||   14.0% |  799.0 |   -- |    -- |line.178
4|||    2.7% |  153.0 |   -- |    -- |line.180
4|||   55.7% | 3183.0 |   -- |    -- |line.181
4|||    1.2% |   66.0 |   -- |    -- |line.211
4|||    1.1% |   60.0 |   -- |    -- |line.212
||||=========================================================
||===========================================================
|   1.3% |   73.0 |   -- |    -- |ETC
||-----------------------------------------------------------
||   1.2% |   71.0 |   -- |    -- |__svml_log4_e9
|============================================================
...
```

From a sampling run

Default sampling interval:
10,000 microsec or 0.01 sec

CrayPat results for a trace run: “Observations and suggestions” section

From a tracing run

```
$ cat myt.rpt
...
D1 cache utilization:

    100.0% of total execution time was spent in 2 functions with D1
    cache hit ratios below the desirable minimum of 75.0%. Cache
    utilization might be improved by modifying the alignment or stride
    of references to data arrays in these functions.

       D1    Time%  Function
     cache
      hit
     ratio

      56.2%    0.0%  main
      70.6%  100.0%  MAIN__
```

```
D1 + D2 cache utilization:

    100.0% of total execution time was spent in 1 functions with
    combined D1 and D2 cache hit ratios below the desirable minimum of
    85.0%. Cache utilization might be improved by modifying the
    alignment or stride of references to data arrays in these functions.

     D1+D2    Time%  Function
     cache
       hit
     ratio

      84.8%  100.0%  MAIN__
```

```
TLB utilization:

    100.0% of total execution time was spent in 2 functions with fewer
    than the desirable minimum of 200 data references per TLB miss. TLB
    utilization might be improved by modifying the alignment or stride
    of references to data arrays in these functions.

     LS per    Time%  Function
     TLB DM

      166.59     0.0%  main
      178.22   100.0%  MAIN__
```

CrayPat results in the text report (2)

```
USER / MAIN__
------------------------------------------------------------------------------
  Time%                                                   100.0%
  Time                                                 54.383497 secs
  Imb. Time                                                   -- secs
  Imb. Time%                                                  --
  Calls                                0.018 /sec            1.0 calls
  CPU_CLK_UNHALTED:THREAD_P                         165268349017
  CPU_CLK_UNHALTED:REF_P                              5165651720
  DTLB_LOAD_MISSES:MISS_CAUSES_A_WALK                  220360299
  DTLB_STORE_MISSES:MISS_CAUSES_A_WALK                  51001827
  L1D:REPLACEMENT                                    14217002486
  L2_RQSTS:ALL_DEMAND_DATA_RD                        11410446231
  L2_RQSTS:DEMAND_DATA_RD_HIT                         4058248212
  MEM_UOPS_RETIRED:ALL_LOADS                         48362710284
  FP_COMP_OPS_EXE:SSE_SCALAR_DOUBLE                       725400
  FP_COMP_OPS_EXE:X87                                 1588974370
  FP_COMP_OPS_EXE:SSE_FP_PACKED_DOUBLE                 209746182
  SIMD_FP_256:PACKED_SINGLE                             34072888
  SIMD_FP_256:PACKED_DOUBLE                          51986600374
  User time (approx)                  54.384 secs   130574801568 cycles  100.0% Time
  CPU_CLK                             3.199GHz
  HW FP Ops / User time             3865.660M/sec   210228176734 ops   20.1%peak(DP)
  Total SP ops                         5.012M/sec      272583104 ops
  Total DP ops                      3860.648M/sec   209955593630 ops
  Computational intensity               1.61 ops/cycle      4.35 ops/ref
  MFLOPS (aggregate)                 3865.66M/sec
  TLB utilization                     178.22 refs/miss     0.348 avg uses
  D1 cache hit,miss ratios             70.6% hits          29.4% misses
  D1 cache utilization (misses)         3.40 refs/miss     0.425 avg hits
  D2 cache hit,miss ratio              48.3% hits          51.7% misses
  D1+D2 cache hit,miss ratio           84.8% hits          15.2% misses
  D1+D2 cache utilization               6.58 refs/miss     0.822 avg hits
  D2 to D1 bandwidth           12806.058MiB/sec   730268558784 bytes
  Average Time per Call                                54.383497 secs
  CrayPat Overhead : Time
==========================
...
```

From a tracing run

Using the default performance counter group value (2) for PAT_RT_PERFCTR (among available values: 0, 1, 2, 6, 7,8, 9, 10, 11, 12, 13, 14 & 19)

SSE scalar double precision (a)
Single or double precision? (b)
SSE vector (128-bit wide) double precision (c)
AVX vector (256-bit wide) single precision (d)
AVX vector (256-bit wide) double precision (e)

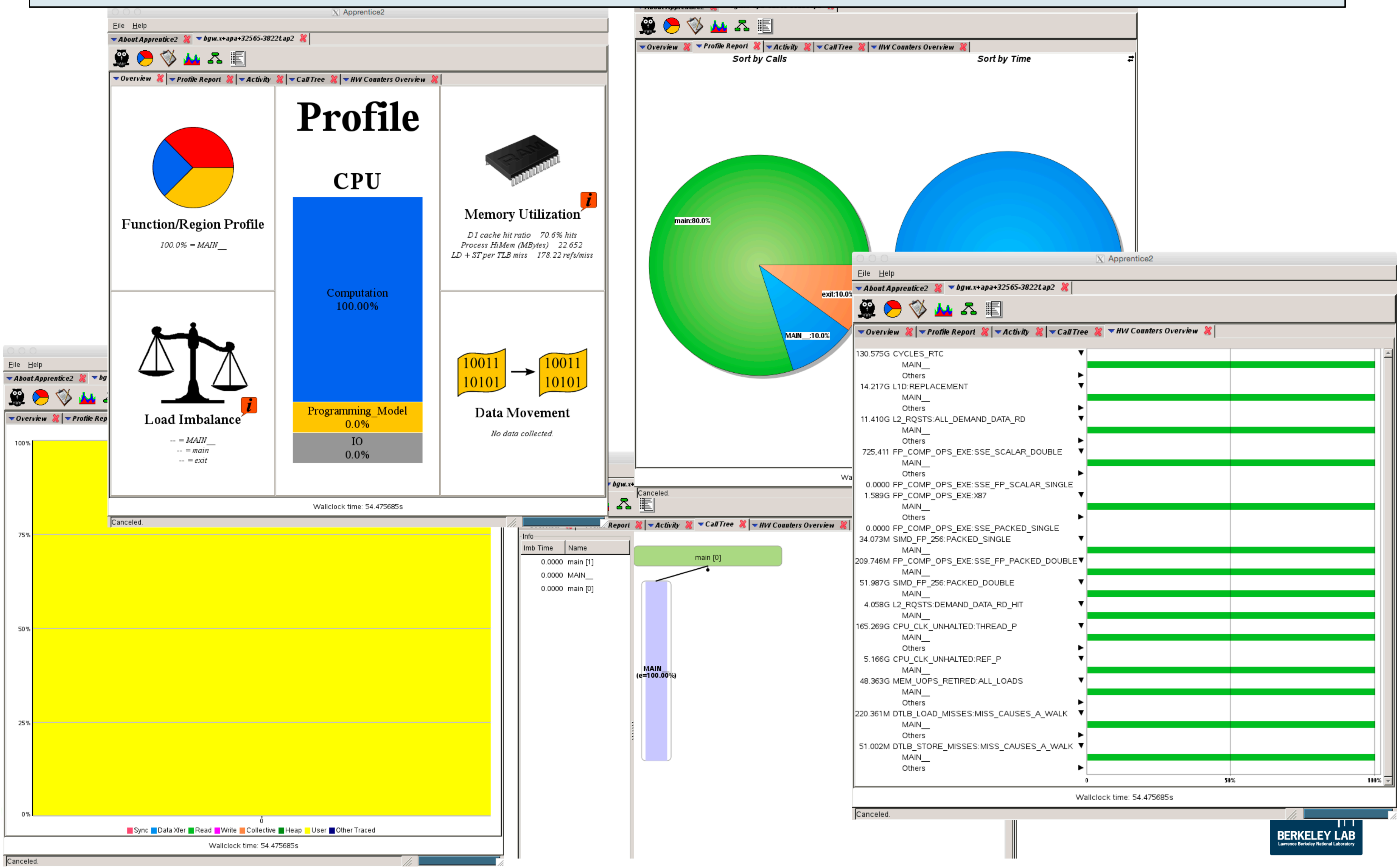
Derived from the above numbers

Average fp vector length
≈ [1*(a)+1*(b)+2*(c)+8*(d)+4*(e)]
/ [(a)+(b)+…+(e)]
= 3.906

See http://icl.cs.utk.edu/projects/papi/wiki/PAPITopics:SandyFlops

CrayPat .ap2 displayed with app2

```
$ app2 bgw.x+apa+32565-38225t.ap2
```



More things to do with CrayPat…

- **Automatic Rank Order Analysis**
  - Suggests a better MPI rank placement
- **CrayPat API**
  - Instrument and get tracing results only for selected regions of your code
- **Monitor a selected group of hardware counters (floating point operations, cache usage, etc.) or network performance counters**
- **For more info:**
  - Man pages: 'intro_craypat', 'craypat-lite', 'pat_build', 'hwpc', 'nwpc', 'pat_report', 'pat_help', 'grid_order', … (after loading the perftools module)
  - Pat_help online help systems
    $ pat_help
  - http://www.nersc.gov/users/software/debugging-and-profiling/craypat/
  - (Old) CrayPat tutorial materials in the directory on NERSC machines:
    /project/projectdirs/training/NUG2012/perftools
  - 'Using Cray Performance Measurement and Analysis Tools', http://docs.cray.com/books/S-2376-622/S-2376-622.pdf
  - 'Using the Aries Hardware Counters', http://docs.cray.com/books/S-0045-10/S-0045-10.pdf
  - 'Using the PAPI Cray NPU Component', http://docs.cray.com/books/S-0046-10/S-0046-10.pdf

CrayPat-lite

- **A simplified version of CrayPat**
  - No need for you to manually build an instrumented binary
  - *.ap2, *.rpt (text report) files are generated for you

```
$ module rm darshan
$ module load perftools-lite
$ export CRAYPAT_LITE=sample_profile
$#export CRAYPAT_LITE=event_profile
$ ftn -c -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -o bgw.x

$ cat runit
...
 aprun -n 1 ./bgw.x
$ qsub runit
2448485.edique02

$ more runit.o2448485
$ more bgw.x+40813-5014s.rpt
$ app2 bgw.x+40813-5014s.ap2
```

Unload darshan as it will interfere with perftools

For sampling (default)

For tracing

'bgw.x' is an instr. binary

Performance summary included in stdout file

Same text report saved in bgw.x+*.rpt

If you want…

Cray Reveal

- **Identifies potential loops for OpenMP parallelization**
  - Based on analysis of CrayPat's performance run
- **Provides OpenMP directives for such loops**
- **Works only with Cray compiler at this time**
- **Additionally, it displays loopmark information in GUI, generated by Cray compiler (i.e., what kind of optimization is done to a loop, whether it is vectorized, how many times it is unrolled, etc.)**
- **See https://www.nersc.gov/users/software/debugging-and-profiling/craypat/reveal/**

Workflow with Reveal

```
$ module rm darshan
$ module swap PrgEnv-intel PrgEnv-cray
$ module load perftools
```

Works only under PrgEnv-cray

(1) Generate loop work estimates

```
$ ftn -c -O3 -h profile_generate bgw.f90
$ ftn -O3 bgw.o -h profile_generate -o bgw.x
$ pat_build -f -w bgw.x
$ cat runit
...
 aprun -n 1 ./bgw.x+pat
$ qsub runit
2450557.edique02
$ pat_report bgw.x+pat+38560-5949t.xf > my.rpt
```

Build an instr. binary for tracing

Get performance data

(2) Generate a "program library"

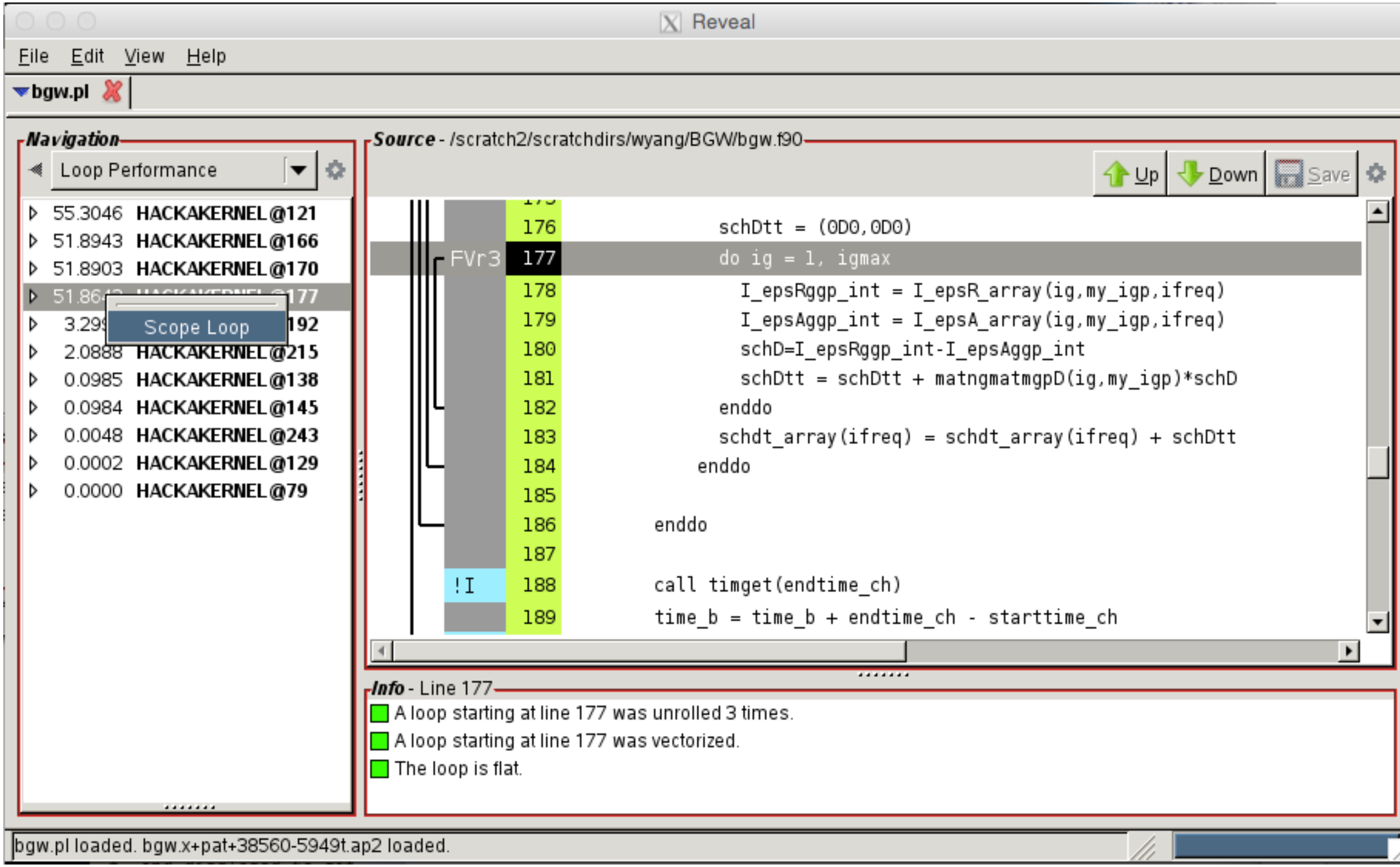
```
$ ftn -c -h pl=bgw.pl -O3 bgw.f90
$ ls -ld bgw.pl
drwx------  2 wyang wyang  4096 Feb 24 22:48 bgw.pl
```

Repeat for all source files

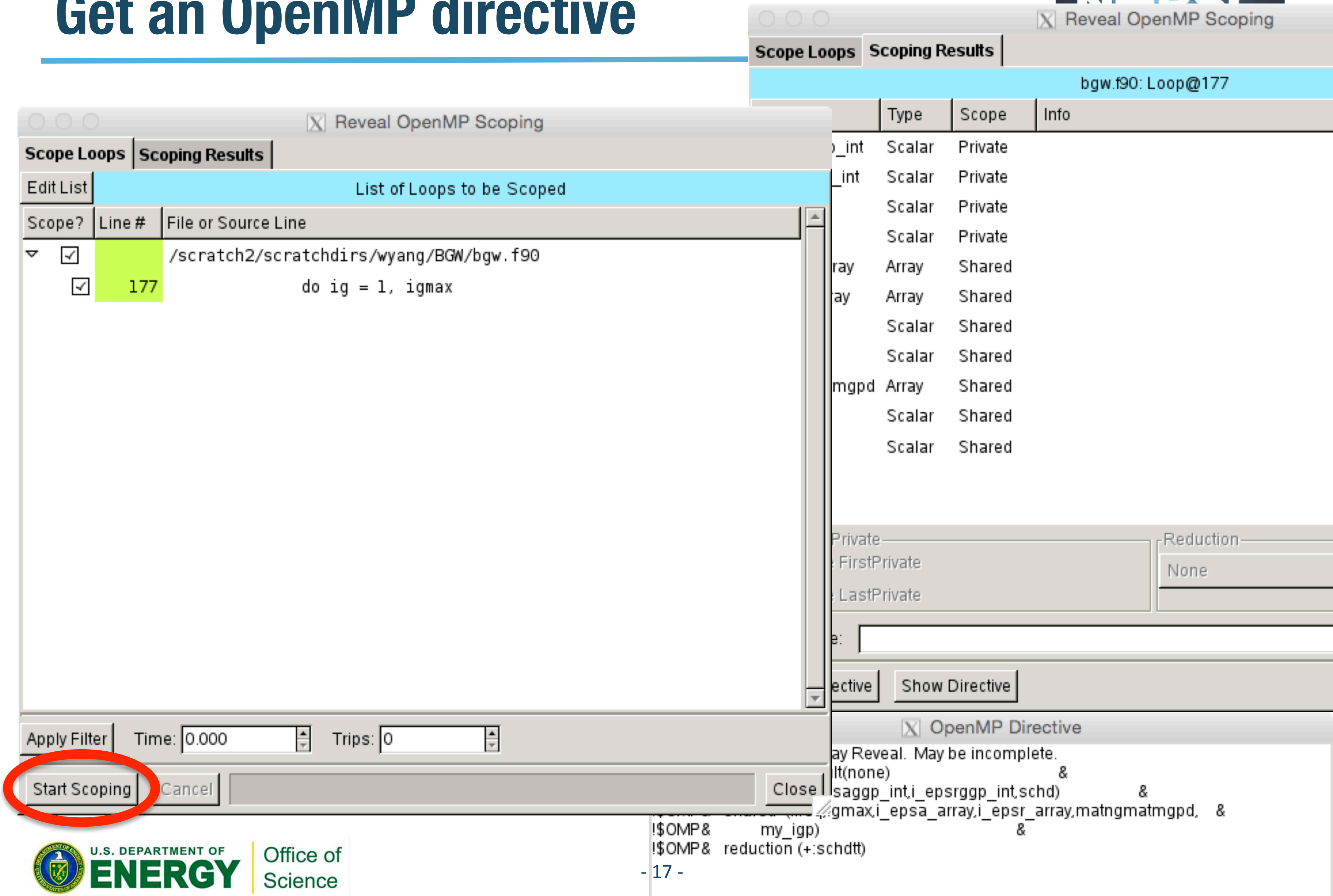
(3) Run Reveal to identify potential loops that can be turned into OpenMP parallel regions

```
$ reveal bgw.pl bgw.x+pat+38560-5949t.ap2
```

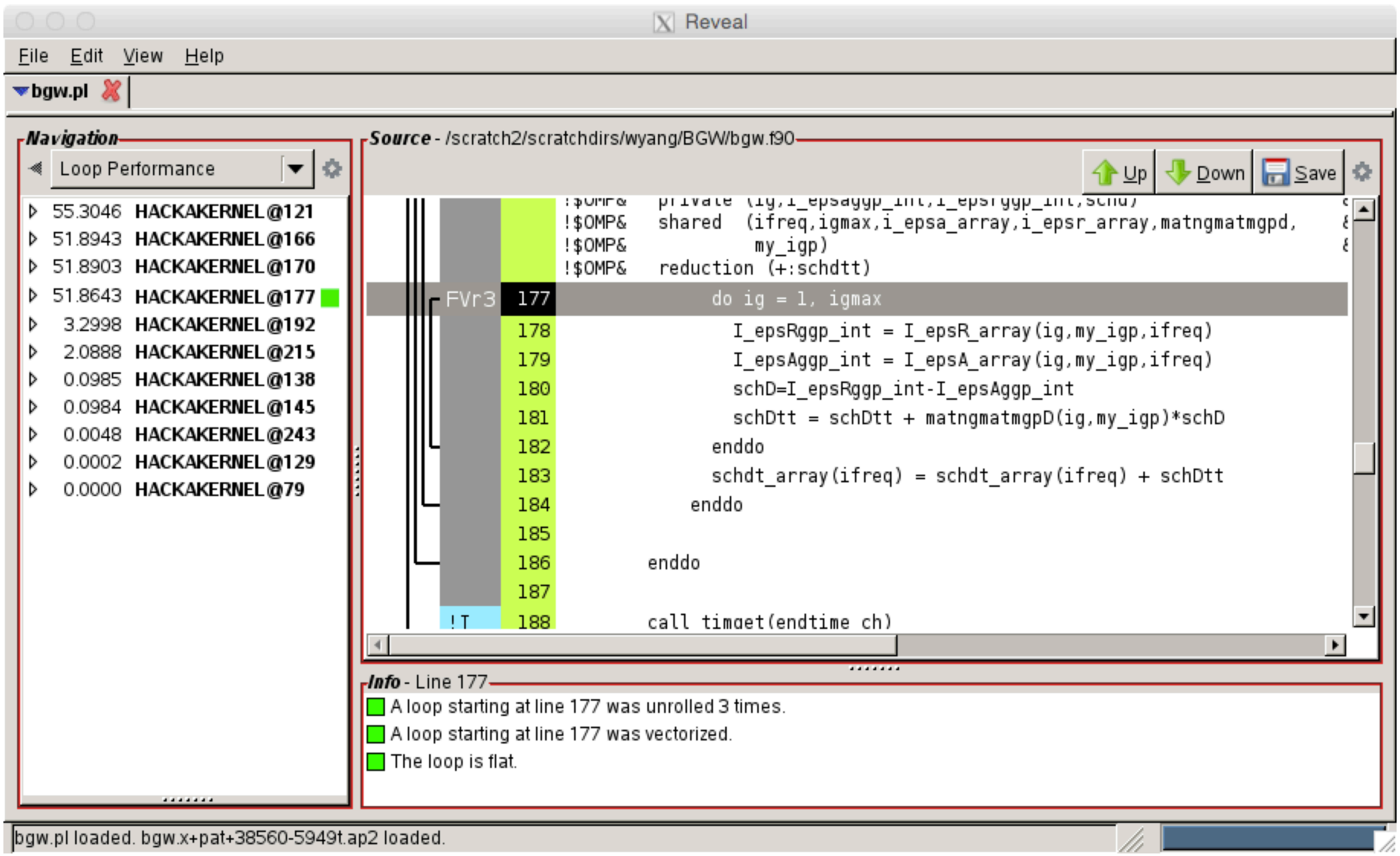
Reveal: scope a loop



Get an OpenMP directive



Add the directive to your code



U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

MAP

- **Allinea's parallel profiling tool with GUI: sampling**
- **MAP licenses for ~ 512 MPI tasks**
  – Shared by other users and among all machines
- **Need to build two small libraries for sampling, MAP sampler and MPI wrapper libraries**
  – make-profile-libraries
  – Need to follow a certain linking order
    - See the user manual for details
    - Usually OK if you follow the instructions printed when running the above commands
- **For info:**
  – $ALLINEA_TOOLS_DOCDIR/userguide.pdf (after loading the allineatools module)
  – https://www.nersc.gov/users/software/debugging-and-profiling/MAP/

Using MAP

```
$ module load allineatools/5.0-40932
$ make-profiler-libraries --lib-type=static
$ ftn -c -g -O3 -xAVX -openmp bgw.f90
$ ftn -O3 -xAVX -openmp bgw.o -Wl,@./allinea-profiler.ld -o bgw.x
```

Build the 2 static libs that MAP needs

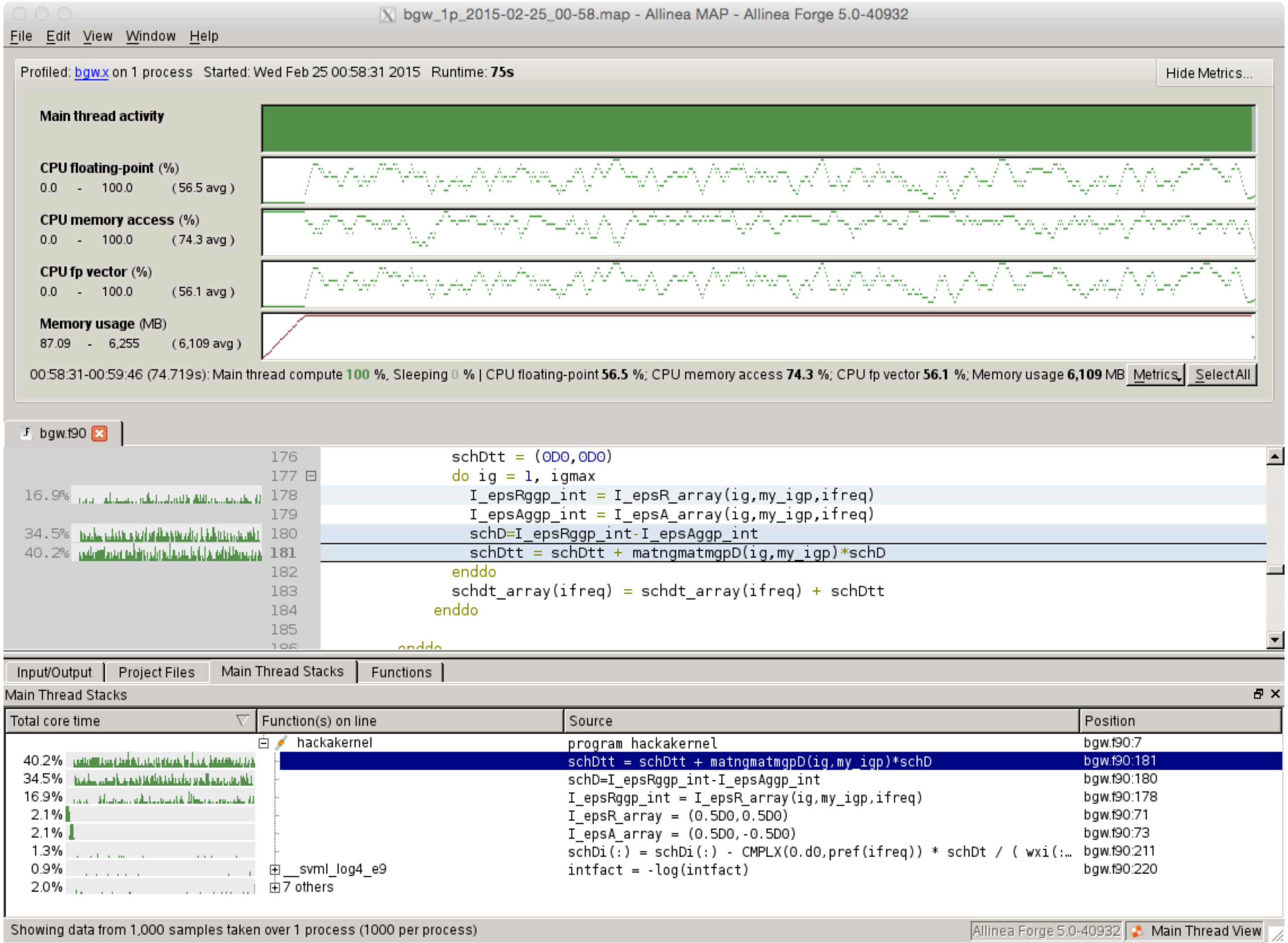
Build using the MAP-generated option file

```
$ qsub -I -v DISPLAY -lmppwidth=24
$ cd $PBS_O_WORKDIR
$ module load allineatools/5.0-40932
$ map ./bgw.x
$ ls -lrt
…
-rw-------  1 wyang wyang    90885 Feb 25 00:59 bgw_1p_2015-02-25_00-58.map
```

Run with MAP

Profiling results saved in a file

MAP results



bgw_1p_2015-02-25_00-58.map - Allinea MAP - Allinea Forge 5.0-40932
File Edit View Window Help
Profiled: bgw.x on 1 process Started: Wed Feb 25 00:58:31 2015 Runtime: 75s
Hide Metrics...
Main thread activity
CPU floating-point (%) 0.0 - 100.0 (56.5 avg)
CPU memory access (%) 0.0 - 100.0 (74.3 avg)
CPU fp vector (%) 0.0 - 100.0 (56.1 avg)
Memory usage (MB) 87.09 - 6,255 (6,109 avg)
00:58:31-00:59:46 (74.719s): Main thread compute 100 %, Sleeping 0 % | CPU floating-point 56.5 %; CPU memory access 74.3 %; CPU fp vector 56.1 %; Memory usage 6,109 MB
Metrics
Select All
bgw.f90
schDtt = (0D0,0D0)
do ig = 1, igmax
I_epsRggp_int = I_epsR_array(ig,my_igp,ifreq)
I_epsAggp_int = I_epsA_array(ig,my_igp,ifreq)
schD=I_epsRggp_int-I_epsAggp_int
schDtt = schDtt + matngmatmgpD(ig,my_igp)*schD
enddo
schdt_array(ifreq) = schdt_array(ifreq) + schDtt
enddo
16.9%
34.5%
40.2%
Input/Output
Project Files
Main Thread Stacks
Functions
Main Thread Stacks
Total core time
Function(s) on line
Source
Position
hackakernel
program hackakernel
bgw.f90:7
40.2%
schDtt = schDtt + matngmatmgpD(ig,my_igp)*schD
bgw.f90:181
34.5%
schD=I_epsRggp_int-I_epsAggp_int
bgw.f90:180
16.9%
I_epsRggp_int = I_epsR_array(ig,my_igp,ifreq)
bgw.f90:178
2.1%
I_epsR_array = (0.5D0,0.5D0)
bgw.f90:71
2.1%
I_epsA_array = (0.5D0,-0.5D0)
bgw.f90:73
1.3%
schDi(:) = schDi(:) - CMPLX(0.d0,pref(ifreq)) * schDt / (wxi(:…
bgw.f90:211
0.9%
__svml_log4_e9
intfact = -log(intfact)
bgw.f90:220
2.0%
7 others
Showing data from 1,000 samples taken over 1 process (1000 per process)
Allinea Forge 5.0-40932
Main Thread View

HPCToolkit

- Supports MPI and threading (OpenMP, pthreads)
- Sampling based
- Assembles performance measurements into a call path profile that associates the costs of each function call with its full calling context
- Your program should call MPI_Comm_rank() with MPI_COMM_WORLD preferably right after MPI_Init(), to let hpcrun know these MPI ranks

HPCToolkit Workflow

1. **Build your code**
   - Compile code with –g
   - For statically linked binary, run hpclink
2. **Run hpcstruct on your executable to analyze code structure**
3. **Run your code with hpcrun**

   Measurement data in hpctoolkit-*app*-measurements-$PBS_JOBID
4. **Run hpcprof/hpcprof-mpi to correlate collected data with the code structure**

   Data base in hpctoolkit-*app*-database-$PBS_JOBID
5. **View the result with hpcviewer or hpctraceviewer**

Events to be sampled

- **Sample sources:**
  - -e *e1@p1* -e *e2@p2* …

  where *e1*, *e2*,… are events and *p1*, *p2*,… are sampling periods
  - Example: -e PAPI_TOT_CYC@15000000 -e PAPI_L2_TCM@400000
  - Alternatively, use the environment variable

  $ export HPCRUN_EVENT_LIST="PAPI_TOT_CYC@15000000;PAPI_L2_TCM@400000"
- **PAPI event must be both available and not derived**
  - Aim for a rate for approx. a few hundred samples per second
    - Several million or tens of millions for PAPI_TOT_CYC
    - A few hundred thousand for cache misses
- **Proxy sampling for derived PAPI events for events that cannot trigger interrupts directly**
  - For example, PAPI_FP_OPS on Intel CPUs
  - Sampling period not specified: e.g., '-e PAPI_FP_OPS'

HPCToolkit

```
% module load hpctoolkit
% ftn -c -g -openmp jacobi_mpiomp.f90
% hpclink ftn -openmp -o jacobi_mpiomp jacobi_mpiomp.o
%#ftn -openmp -dynamic -o jacobi_mpiomp jacobi_mpiomp.o
```

Statically-linked binary
Dynamically-linked binary

```
% hpcstruct jacobi_mpiomp
```

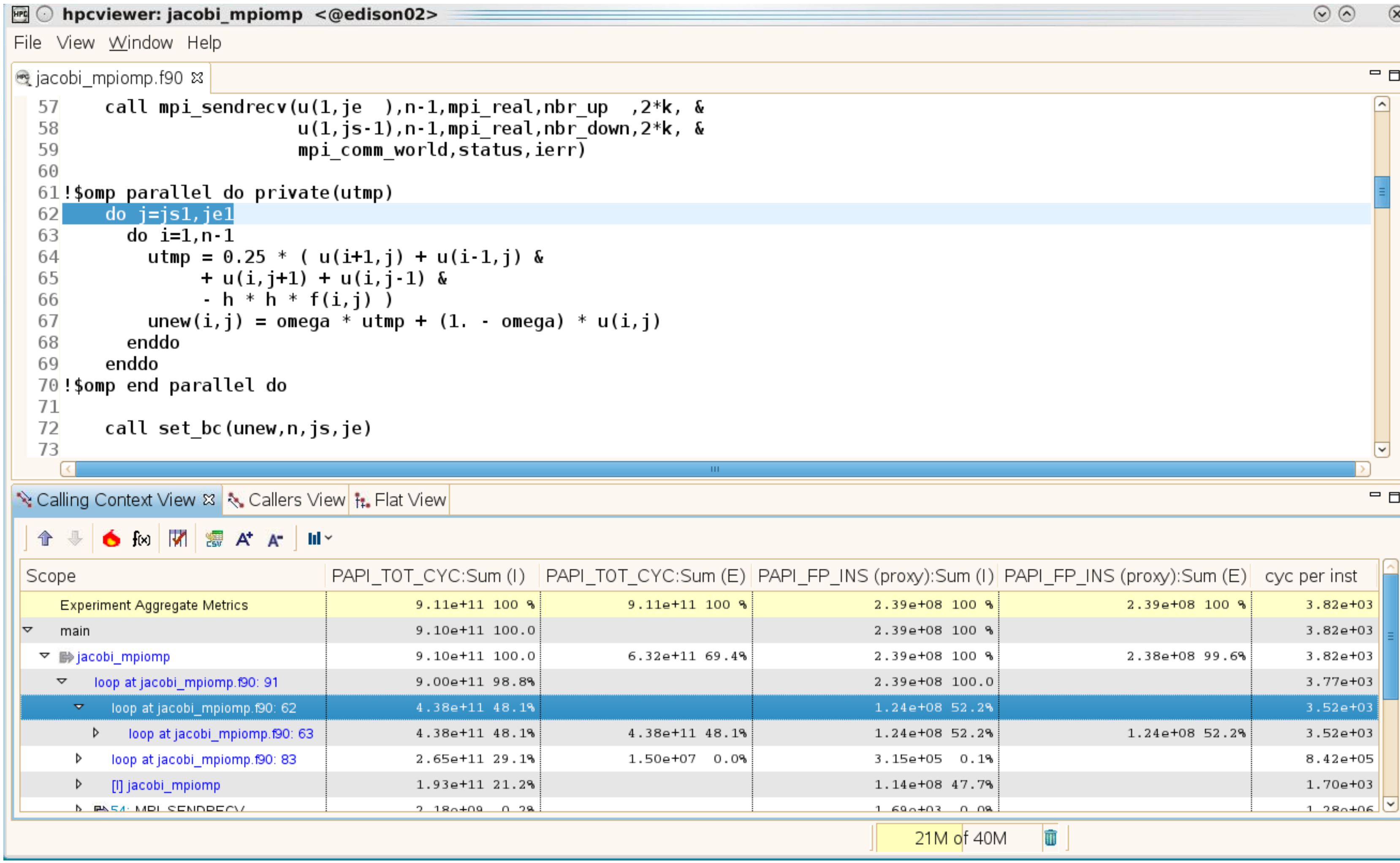
Analyze code in order to map collected data; will create jacobi_mpiomp.hpcstruct

```
% cat runit
…
 module load hpctoolkit
 export OMP_NUM_THREADS=12
 mdir=res_m.$PBS_JOBID
 ddir=res_d.$PBS_JOBID
 aprun -n 4 -N 2 -S 1 -d 12 hpcrun –e PAPI_L2_TCM@10000 ... -o $mdir \
       ./jacobi_mpiomp
 aprun -n 4 -N 2 -S 1 -d 12 hpcprof-mpi -S jacobi_mpiomp.hpcstruct -I $PWD/'*' \
       -o $ddir $mdir
```

Directory for measurement data
Directory for database

```
% qsub runit
2442996.edique02
% hpcviewer res_d.2442996.edique02
```

View the results


hpcviewer: jacobi_mpiomp <@edison02>
File View Window Help
jacobi_mpiomp.f90
call mpi_sendrecv(u(1,je),n-1,mpi_real,nbr_up ,2*k, &
u(1,js-1),n-1,mpi_real,nbr_down,2*k, &
mpi_comm_world,status,ierr)
!$omp parallel do private(utmp)
do j=js1,je1
do i=1,n-1
utmp = 0.25 * (u(i+1,j) + u(i-1,j) &
+ u(i,j+1) + u(i,j-1) &
- h * h * f(i,j))
unew(i,j) = omega * utmp + (1. - omega) * u(i,j)
enddo
enddo
!$omp end parallel do
call set_bc(unew,n,js,je)
Calling Context View
Callers View
Flat View
Scope
PAPI_TOT_CYC:Sum (I)
PAPI_TOT_CYC:Sum (E)
PAPI_FP_INS (proxy):Sum (I)
PAPI_FP_INS (proxy):Sum (E)
cyc per inst
Experiment Aggregate Metrics 9.11e+11 100 % 9.11e+11 100 % 2.39e+08 100 % 2.39e+08 100 % 3.82e+03
main 9.10e+11 100.0 2.39e+08 100 % 3.82e+03
jacobi_mpiomp 9.10e+11 100.0 6.32e+11 69.4% 2.39e+08 100 % 2.38e+08 99.6% 3.82e+03
loop at jacobi_mpiomp.f90: 91 9.00e+11 98.8% 2.39e+08 100.0 3.77e+03
loop at jacobi_mpiomp.f90: 62 4.38e+11 48.1% 1.24e+08 52.2% 3.52e+03
loop at jacobi_mpiomp.f90: 63 4.38e+11 48.1% 4.38e+11 48.1% 1.24e+08 52.2% 1.24e+08 52.2% 3.52e+03
loop at jacobi_mpiomp.f90: 83 2.65e+11 29.1% 1.50e+07 0.0% 3.15e+05 0.1% 8.42e+05
[I] jacobi_mpiomp 1.93e+11 21.2% 1.14e+08 47.7% 1.70e+03
21M of 40M

National Energy Research Scientific Computing Center

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory