

CRAY

**Cray Performance Tools Enhancements
for Next Generation Systems**
Heidi Poxon

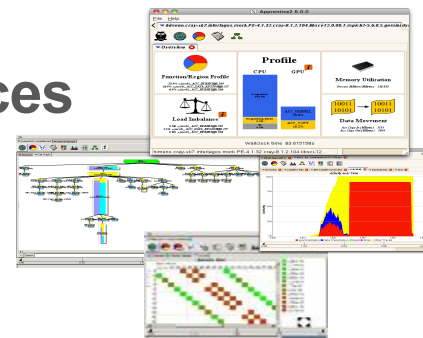


Agenda

- **Cray Performance Tools Overview**
- **Recent Enhancements**
- **Support for Cray systems with KNL**

Cray Performance Analysis Tools Overview

- **Assist the user with application performance analysis and optimization**
 - Help user identify important and meaningful information from potentially massive data sets
 - Help user identify problem areas instead of just reporting data
 - Bring optimization knowledge to a wider set of users
- **Focus on ease of use and intuitive user interfaces**
 - Automatic program instrumentation
 - Automatic analysis
- **Whole program analysis across many nodes**



Two Modes of Use

- **CrayPat-lite** for novice users, or convenience
- **CrayPat** for in-depth performance investigation and tuning assistance
- **Both offer:**
 - Whole program analysis across many nodes
 - Indication of causes of problems
 - Suggestions of modifications for performance improvement

“Lite” Mode

Load performance tools modules

```
> module load perftools-lite
```

Build program

(no modification to makefile)

```
> make
```



```
a.out (instrumented program)
```

Run program

(no modification to batch script)

```
> aprun a.out
```



```
Condensed report to stdout
a.out*.rpt (same as stdout)
a.out*.ap2
files
```

COMPUTE

STORE

Example CrayPat-lite Output



```
#####
#
#           CrayPat-lite Performance Statistics           #
#
#####

CrayPat/X:  Version 6.3.2.461 Revision 56930ff 02/01/16 15:31:33
Experiment:                lite lite/sample_profile
Number of PEs (MPI ranks): 64
Numbers of PEs per Node:  32 PEs on each of 2 Nodes
Numbers of Threads per PE: 1
Number of Cores per Socket: 16
Execution start time: Tue Feb 2 18:53:50 2016
System name and speed: kay 2301 MHz (approx)

Avg Process Time:          64.38 secs
High Memory:               1,563 MBytes 24.43 MBytes per PE
MFLOPS:                   Not supported (see observation below)
I/O Read Rate:            48.514130 MBytes/sec
I/O Write Rate:           22.281350 MBytes/sec
Avg CPU Energy:           41,820 joules 20,910 joules per node
Avg CPU Power:            649.53 watts 324.77 watts per node
```

Table 1: Profile by Function Group and Function (top 10 functions shown)

Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function
				PE=HIDE
100.0%	6,156.2	--	--	Total

66.3%	4,082.5	--	--	USER

11.8%	729.2	48.8	6.4%	mult_su3_mat_vec_sum_4dir
10.2%	629.4	49.6	7.4%	mult_adj_su3_mat_4vec
6.1%	377.1	28.9	7.2%	mult_su3_nn
5.9%	365.4	42.6	10.6%	mult_su3_na
5.4%	329.4	37.6	10.4%	scalar_mult_add_lathwvec_proj
3.8%	232.9	39.1	14.6%	mult_su3_sitelink_lathwvec
=====				
25.3%	1,557.0	--	--	MPI

12.8%	789.3	163.7	17.5%	MPI_wait
6.7%	411.9	74.1	15.5%	MPI_Isend
4.9%	300.2	95.8	24.6%	MPI_Allreduce
=====				
5.9%	365.4	44.6	11.1%	STRING

5.9%	365.4	44.6	11.1%	memcpy
=====				

Guidance: How Can I Learn More?

MPI utilization:

The time spent processing MPI communications is relatively high. Functions and callsites responsible for consuming the most time can be found in the table generated by `pat_report -O callers+src` (within the MPI group).

Guidance: Reduce Shared Resource Contention

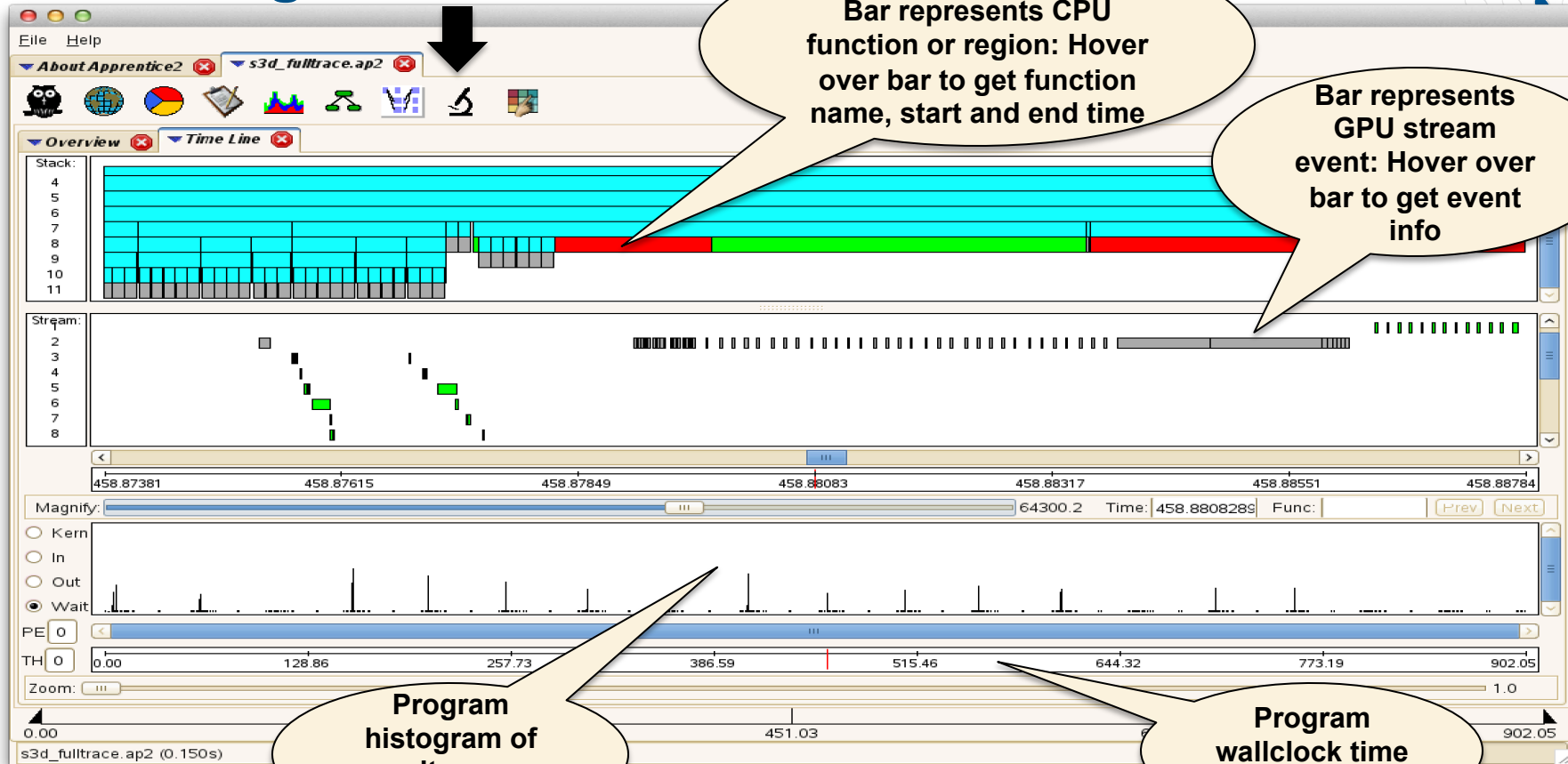
Metric-Based Rank Order:

When the use of a shared resource like memory bandwidth is unbalanced across nodes, total execution time may be reduced with a rank order that improves the balance.

A file named `MPICH_RANK_ORDER.USER_Time` was generated along with this report and contains usage instructions and the custom rank order from the following table.

Rank Order	Node Metric	Reduction in Imb. Value	Maximum Value	Average Value
Current	15.46%		1.134e+03	9.588e+02
Custom	1.46%	14.202%	9.731e+02	9.588e+02

GPU Program Timeline



Program histogram of wait, copy kernel time

Program wallclock time line

Recent Enhancements through perftools/6.3.2

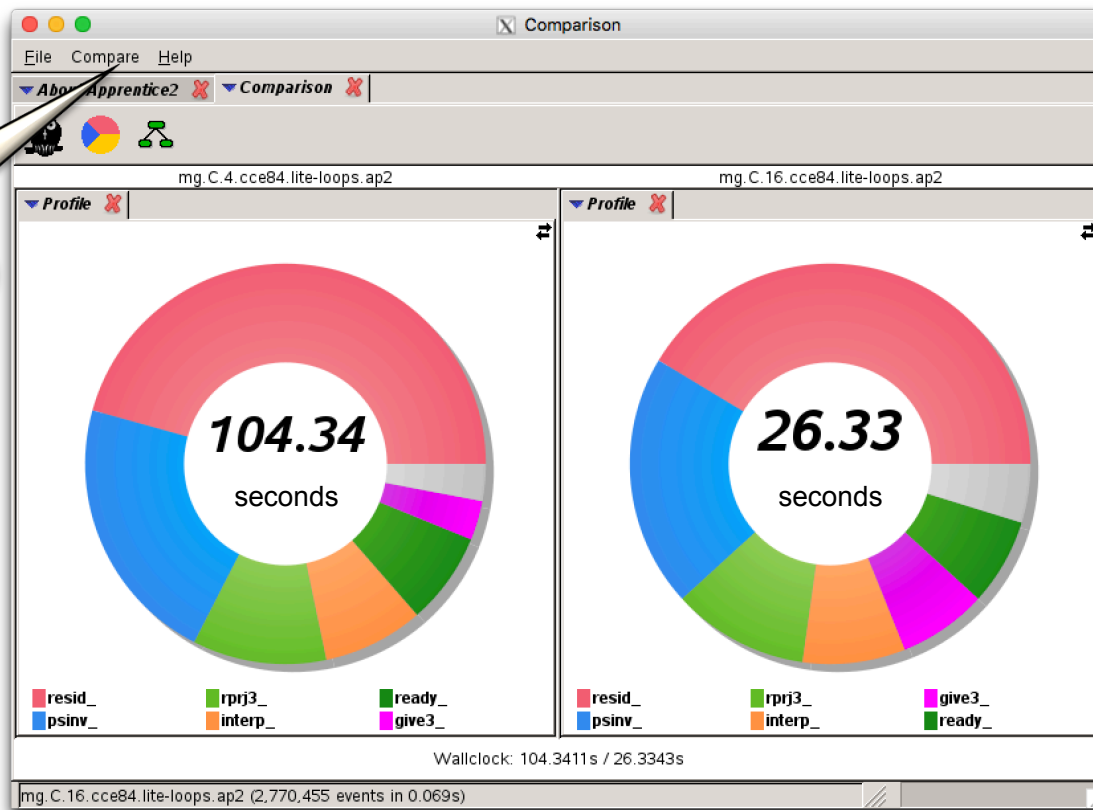
Highlights Since Last CUG

- **New perftools-base and instrumentation modules (6.3.0)**
- **Sampling over time and gnuplots (6.2.3)**
- **Apprentice2 sampling over time plots with call stack (6.3.0)**
- **Apprentice2 MPI communication pattern in summary mode (6.3.0)**
- **Observation for helper threads in reports (6.3.0)**
- **Performance data comparison in Apprentice2 (6.3.1)**

Evaluate Scaling with Cray Apprentice2



Add file for performance comparison here



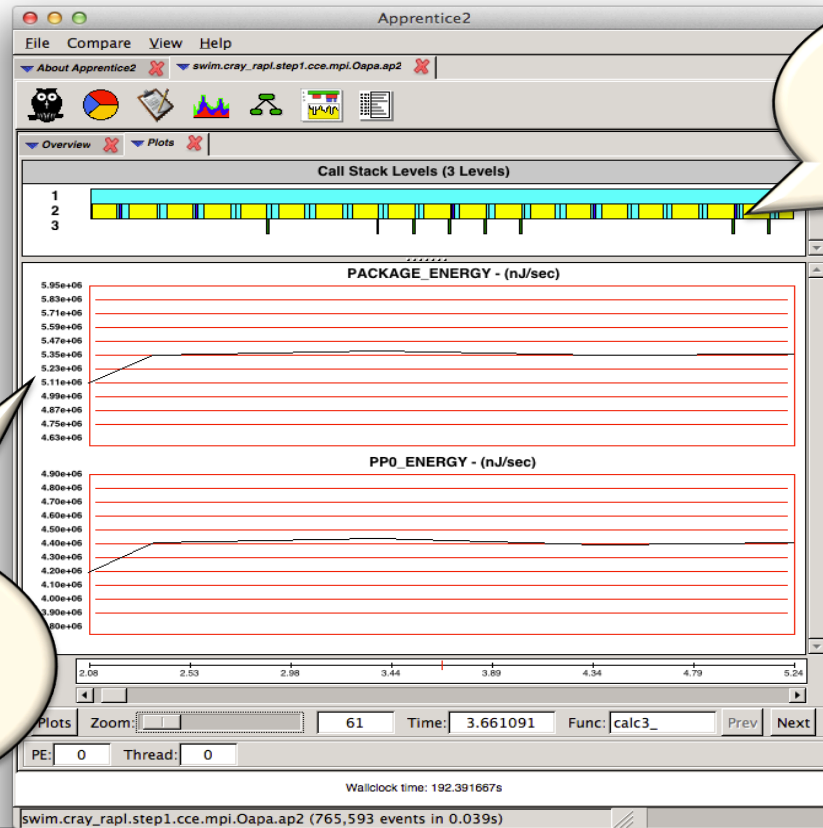
COMPUTE

STORE

ANALYZE

Copyright 2016 Cray Inc.

Energy Consumption Over Time (XC Systems)

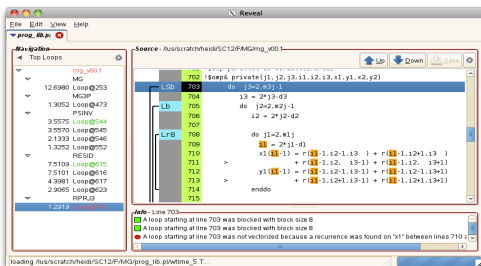


Call stack:
Bar represents function or region: Hover over bar to get function name, start and end time

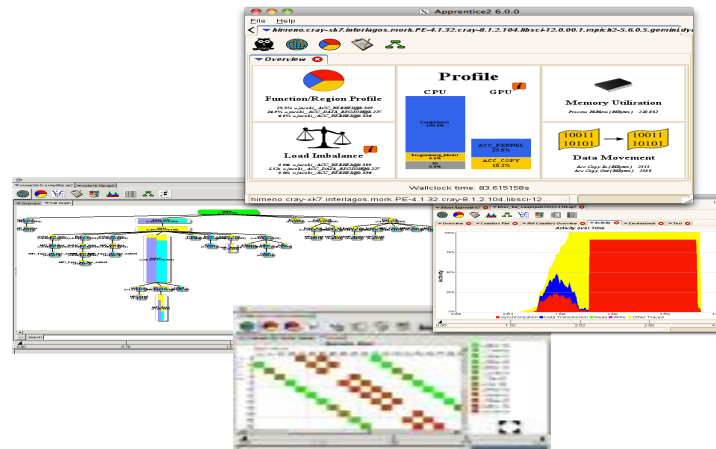
Plots of energy consumed by the socket and by the cores within a socket over time. Can also show memory high water mark, etc.

Support for Cray Systems with KNL

Port



Analyze



Overview of Support for KNL

- **CrayPat and CrayPat-lite**
 - Identifies top time consuming routines, work load imbalance, MPI rank placement strategies, etc.
 - **Enhanced program memory high water mark**
 - Broken down into DDR and MCDRAM memory
 - Report active allocations at samples or during tracing at the function level
- **PAPI**
- **Cray Apprentice2**
 - Helps identify load imbalance, excessive communication, network contention, excessive serialization
- **Reveal support for adding OpenMP and allocating in MCDRAM**

Functionality Coming in 2016

- **MCDRAM configuration information**
- **New trace groups for**
 - MemKind, HBW, CrayMem
 - OpenCL
 - Lustre API
 - Parallel NetCDF
- **Support for Charm++**

Example: MCDRAM Configuration Information

CrayPat/X: Version 6.4.X Revision e82c848 04/29/16 14:13:55

Number of PEs (MPI ranks): 64

Numbers of PEs per Node: 1 PE on each of 64 Nodes

Numbers of Threads per PE: 1

...

Execution start time: Mon May 2 15:54:21 2016

Intel knl CPU ...

MCDRAM: 16 GiB available as snc2, cache (100% cache) for 16 PEs

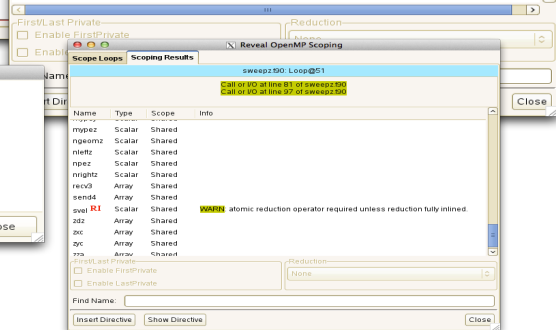
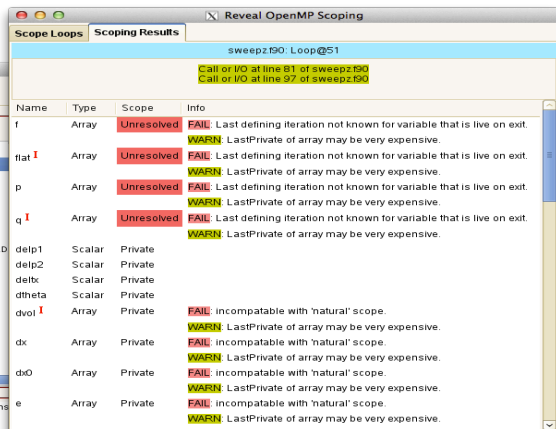
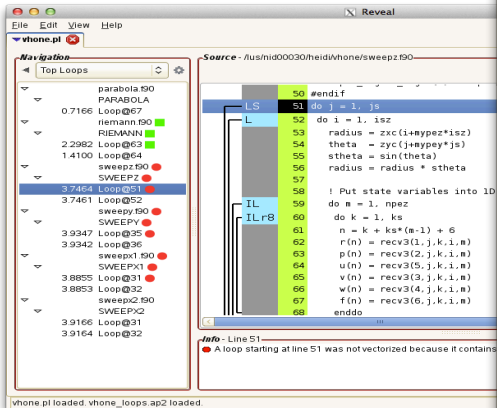
MCDRAM: 16 GiB available as snc2, flat (0% cache) for 16 PEs

MCDRAM: 16 GiB available as snc4, flat (0% cache) for 1 PE

MCDRAM: 16 GiB available as quad, flat (0% cache) for 15 PEs

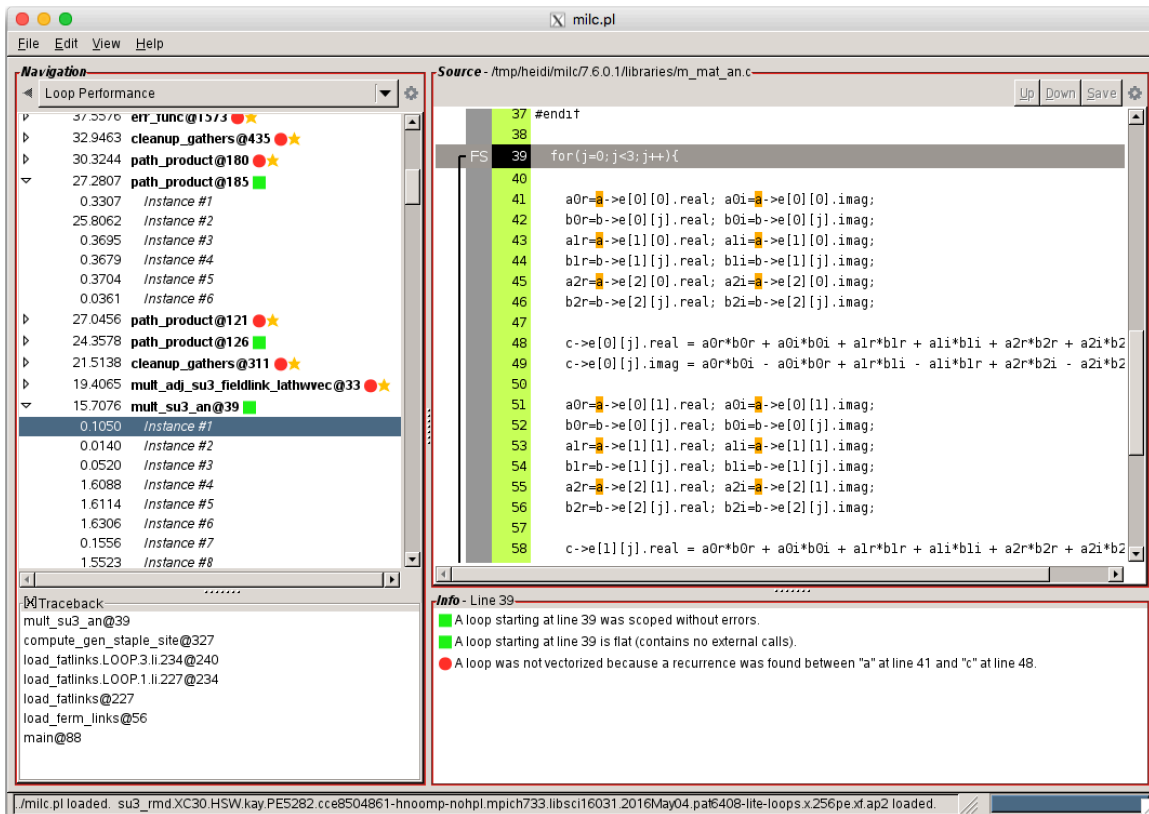
MCDRAM: 16 GiB available as quad, equal (50% cache) for 16 PEs

Adding OpenMP with Reveal



- Navigate to relevant loops to parallelize
- Identify parallelization and scoping issues
- Get feedback on issues down the call chain (shared reductions, etc.)
- Insert parallel directives into source for performance portable code
- Validate scoping correctness on existing directives

More Information for C/C++ Programs



The screenshot shows the milc.pl IDE interface. On the left, the 'Navigation' pane displays a 'Loop Performance' table with columns for time, loop name, and status. The selected loop is 'mult_su3_an@39'. The main window shows the source code for 'Source - /tmp/heid/milc7.6.0.1/libraries/m_mat_an.c', with lines 37-58 visible. The code includes a loop starting at line 39, with vectorization status markers (FS) and comments. The bottom pane shows a 'Traceback' and an 'Info' section with analysis results.

Time	Loop Name	Status
32.9463	cleanup_gathers@435	★
30.3244	path_product@180	★
27.2807	path_product@185	■
0.3307	Instance #1	
25.8062	Instance #2	
0.3695	Instance #3	
0.3679	Instance #4	
0.3704	Instance #5	
0.0361	Instance #6	
27.0456	path_product@121	★
24.3578	path_product@126	■
21.5138	cleanup_gathers@311	★
19.4065	mult_adj_su3_fieldlink_lathwvec@33	★
15.7076	mult_su3_an@39	■
0.1050	Instance #1	
0.0140	Instance #2	
0.0520	Instance #3	
1.6088	Instance #4	
1.6114	Instance #5	
1.6306	Instance #6	
0.1556	Instance #7	
1.5523	Instance #8	

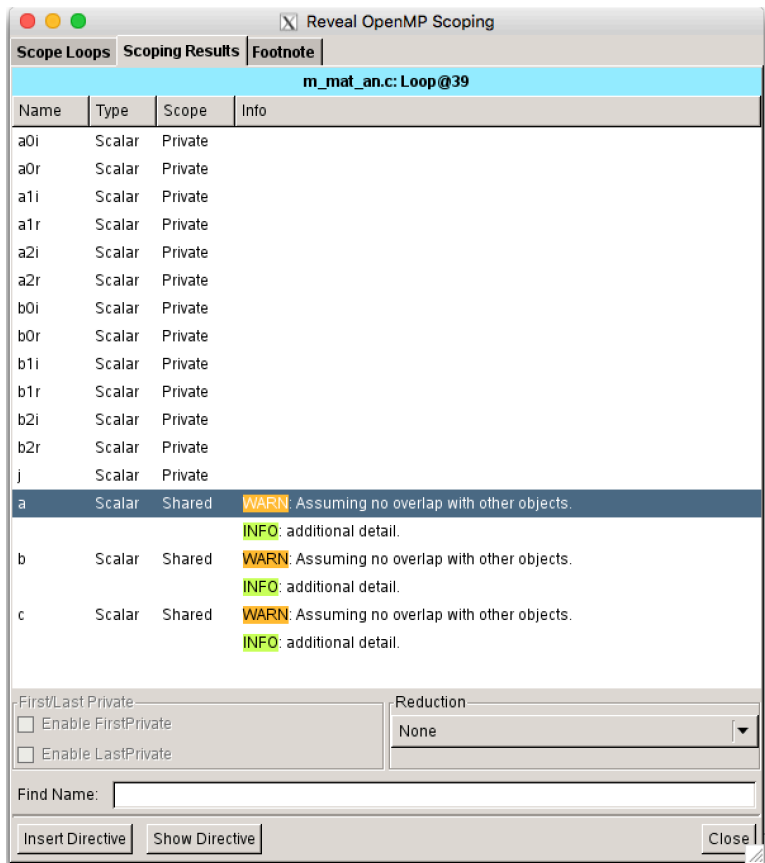
```
#endif
38
39 for(j=0; j<3; j++){
40
41 a0r->e[0][0].real; a0i->e[0][0].imag;
42 b0r-b->e[0][j].real; b0i-b->e[0][j].imag;
43 a1r->e[1][0].real; a1i->e[1][0].imag;
44 b1r-b->e[1][j].real; b1i-b->e[1][j].imag;
45 a2r->e[2][0].real; a2i->e[2][0].imag;
46 b2r-b->e[2][j].real; b2i-b->e[2][j].imag;
47
48 c->e[0][j].real = a0r*b0r + a0i*b0i + a1r*b1r + a1i*b1i + a2r*b2r + a2i*b2i
49 c->e[0][j].imag = a0r*b0i - a0i*b0r + a1r*b1i - a1i*b1r + a2r*b2i - a2i*b2r
50
51 a0r->e[0][1].real; a0i->e[0][1].imag;
52 b0r-b->e[0][j].real; b0i-b->e[0][j].imag;
53 a1r->e[1][1].real; a1i->e[1][1].imag;
54 b1r-b->e[1][j].real; b1i-b->e[1][j].imag;
55 a2r->e[2][1].real; a2i->e[2][1].imag;
56 b2r-b->e[2][j].real; b2i-b->e[2][j].imag;
57
58 c->e[1][j].real = a0r*b0r + a0i*b0i + a1r*b1r + a1i*b1i + a2r*b2r + a2i*b2i
```

Info - Line 39

- A loop starting at line 39 was scoped without errors.
- A loop starting at line 39 is flat (contains no external calls).
- A loop was not vectorized because a recurrence was found between "a" at line 41 and "c" at line 48.

Traceback:
mult_su3_an@39
compute_gen_staple_site@327
load_fatlinks.LOOP3.ll.234@240
load_fatlinks.LOOP1.ll.227@234
load_fatlinks@227
load_ferm_links@56
main@88

More Information for C/C++ Programs (2)



Reveal OpenMP Scoping

Scope Loops | Scoping Results | Footnote

m_mat_an.c: Loop@39

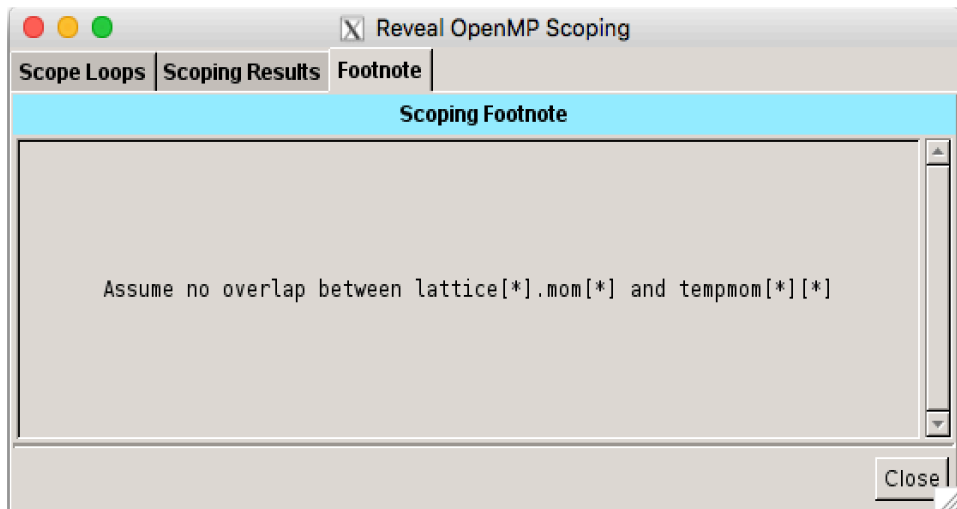
Name	Type	Scope	Info
a0i	Scalar	Private	
a0r	Scalar	Private	
a1i	Scalar	Private	
a1r	Scalar	Private	
a2i	Scalar	Private	
a2r	Scalar	Private	
b0i	Scalar	Private	
b0r	Scalar	Private	
b1i	Scalar	Private	
b1r	Scalar	Private	
b2i	Scalar	Private	
b2r	Scalar	Private	
j	Scalar	Private	
a	Scalar	Shared	WARN: Assuming no overlap with other objects. INFO: additional detail.
b	Scalar	Shared	WARN: Assuming no overlap with other objects. INFO: additional detail.
c	Scalar	Shared	WARN: Assuming no overlap with other objects. INFO: additional detail.

First/Last Private: Enable FirstPrivate Enable LastPrivate

Reduction: None

Find Name:

Insert Directive Show Directive Close



Reveal OpenMP Scoping

Scope Loops | Scoping Results | Footnote

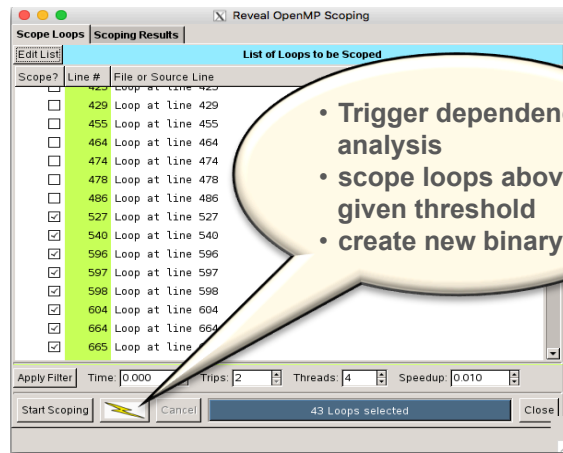
Scoping Footnote

Assume no overlap between lattice[*].mom[*] and tempmom[*][*]

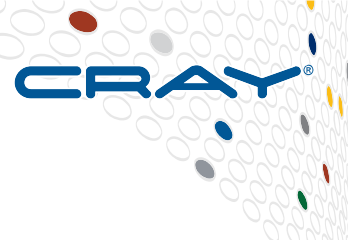
Close

Reveal Auto-Parallelization

- Build an *experimental binary* that includes automatic runtime-assisted parallelization
- No source code changes required to see if high level loops that contain calls can be automatically parallelized
- Result includes parallelization of serial loops via traditional OpenMP as well as more extensive loop optimizations
- User Workflow:
 1. Obtain **loop work estimates** using CCE 8.5 and perftools-lite-loops from perftools/6.4.0
 2. Use Reveal and CCE's program library to **parallelize loops and create experimental binary**
 3. Run experimental binary and compare performance against baseline
 4. If auto-parallelization is successful, **use Reveal to insert parallel directives into source**



Examples of Reveal Analysis Feedback



Navigation

- Loop Performance
- 12.5484 MG_MPI@245
- 6.5747 MG_MPI@664
- 0.1878 Instance #1
- 0.2593 Instance #2
- 0.0130 Instance #3
- 2.8343 Instance #4
- 2.8454 Instance #5
- 0.1455 Instance #6
- 0.1447 Instance #7
- 0.1448 Instance #8
- 6.5736 MG_MPI@665
- 3.8584 MG_MPI@666
- 2.8689 MG_MPI@596
- 2.8683 MG_MPI@597
- 2.5819 MG_MPI@672
- 1.8738 MG_MPI@1069
- 1.7607 MG_MPI@598
- 1.5160 MG_MPI@540
- 1.4182 MG_MPI@527
- 1.0593 MG_MPI@834
- 1.0590 MG_MPI@835
- 1.0583 MG_MPI@750
- 1.0580 MG_MPI@753
- 1.0363 MG_MPI@604
- 0.9935 MG_MPI@1154

Source - /home/heidi/demos/NPB/NPB3.2-MPI-mg/MG/mg.f

Fg	663	do i3=2,n3-1
F	665	do i2=2,n2-1
FVr4	666	do il=1,n1
	667	u1(il) = u(il,i2-1,i3) + u(il,i2+1,i3)
	668	> + u(il,i2,i3-1) + u(il,i2,i3+1)
	669	u2(il) = u(il,i2-1,i3-1) + u(il,i2+1,i3-1)
	670	> + u(il,i2-1,i3+1) + u(il,i2+1,i3+1)
	671	enddo
FVr4	672	do il=2,n1-1
	673	r(il,i2,i3) = u(il,i2,i3)

Info - Line 664

- A loop starting at line 664 was scoped without errors.
- A loop starting at line 664 is flat (contains no external calls).
- A loop starting at line 664 was not vectorized because a recurrence was found on "u1" between lines 667 and 670.
- A loop starting at line 664 was partitioned.
- A loop starting at line 665 is flat (contains no external calls).
- A loop starting at line 665 was not vectorized because a recurrence was found on "u1" between lines 667 and 670.
- A loop starting at line 666 is flat (contains no external calls).
- A loop starting at line 666 was unrolled 4 times.
- A loop starting at line 666 was vectorized.
- A loop starting at line 672 is flat (contains no external calls).
- A loop starting at line 672 was unrolled 4 times.
- A loop starting at line 672 was vectorized.

Reveal OpenMP Scoping

Scope Loops | Scoping Results | Build Results

Your binary was rebuilt with the following changes.

- /home/heidi/demos/NPB/NPB3.2-MPI-mg/MG/mg.f
 - OMP loop at line 596
 - OMP loop at line 664
 - OMP loop at line 750
 - OMP loop at line 834
 - OMP loop at line 995
 - OMP loop at line 1154
 - Autothreaded loop at line 1199
 - Autothreaded loop at line 1213
 - Autothreaded loop at line 1262
 - Autothreaded loop at line 1276
 - Autothreaded loop at line 1326
 - Autothreaded loop at line 1335
 - Autothreaded loop at line 1370
 - Autothreaded loop at line 1379
 - OMP loop at line 2173
 - OMP loop at line 2435



Reveal Auto-Parallelization Recap

- **Minimal user time investment includes time to set up and run optimization experiment**
 - Collect loop work estimates
 - Build program library
 - Click button in Reveal
 - Run experimental binary and compare against original program
- **Even if experiment does not yield a performance improvement, Reveal will provide insight into parallelization issues**
- **Targeted for KNL, where a pure MPI solution cannot utilize all cores on a node**
- **Can be used on existing hardware (AMD Interlagos, Intel Haswell, etc.)**
- **Infrastructure will allow different optimization experiments in the future**

Coming Soon...

Identifying Objects for Allocation in KNL MCDRAM

FLOPS vs Data Movement / Data Locality

- **Next generation memory systems will be more complicated**
 - Multi-tier hierarchies
 - NUMA domains
 - Complicated caches
- **Data movement through the memory hierarchy is critical to performance**
- **Compared to the price of data movement, flops are “free”**
- **Monitoring and minimizing data movement within node and across nodes is key**

Identifying Objects for Allocation in MCDRAM



- Is application predominantly memory intensive?
- Does application data all fit within MCDRAM?
- What data contributes most to memory bandwidth?
- Where is data allocated within program?

Reveal Data Allocation Assistance

- Use combination of CrayPat and Reveal to identify data most relevant for allocation in MCDRAM

```
subroutine sweep (it,jt, kt, nm, isct, mm,mmo,mmi, mk, myid,  
 1 hi, hj, hk, di, dj, dk, Phi, Phii,  
 2 Src, Flux, Sigt,  
 3 w,mu,eta,tsi, wmu,weta,wtsi, pn,  
...  
387   do i = 1, it  
388     phi(i) = src(i,j,k,1)  
389   end do  
390   do n = 2, nm  
391     do i = 1, it  
392       phi(i) = phi(i) + pn(m,n,iq)*src(i,j,k,n)  
393     end do  
394   end do
```

```
c...AUTOMATIC ARRAYS  
double precision Src(it,jt,kt,nm)  
double precision pn(mm,nm,8)  
double precision Phi(it)
```



Example Workflow

- **Enable data collection for memory analysis experiment and build program with CCE's program library feature**
- **Run program to collect data**
- **Pass program library and memory traffic data to Reveal**
- **Reveal shows best candidates for allocation in MCDRAM and identifies points of allocation**
- **Reveal helps insert allocation directives into source**

Summary

- **Users continue to need tools to help find critical performance bottlenecks within a program**
- **Cray performance tools offer functionality that reduces the time investment associated with porting and tuning applications on new and existing Cray systems**

Q&A

Heidi Poxon
heidi@cray.com

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.