THE SUPERCOMPUTER COMPANY

An Update on the Cray Tools Activities for Extreme Scale Computing

Luiz DeRose and Heidi Poxon
Cray Inc.

Moving from X86 to Hybrid Multi-core Systems

- Running MPI only on a node will not work well
  - Too much memory used, even if on-node shared communication is available
  - As the number of MPI ranks increases, more off-node communication can result, creating a network injection issue

- Focus on where MPI starts leveling off

- Address by adding additional levels of parallelism, reducing MPI ranks per node
  - MPI -> MPI + OpenMP
  - MPI + OpenMP -> MPI + OpenMP GPU extensions

Steps to Porting to Hybrid Multi-core Systems

- Maximize on-node communication if MPI point-to-point communication is dominant in the program
  - Auto-grid detection and placement suggestions

- Determine where to add additional levels of parallelism
  - Find top time consuming loops with enough work for GPU
    - Loop statistics

- Do parallel analysis and restructuring on targeted high level loops
  - Scoping assistance

Steps to Porting to Hybrid Multi-core Systems (2)

- Add parallel directives and acceleration extensions
  - OpenMP extensions
- Run on X86 + GPU and get performance feedback
- Optimize for data locality and copies to the GPU
- Optimize kernel on GPU
  - Cray performance tools statistics

Automatic Communication Grid Detection

- Analyze runtime performance data to identify grids in a program to maximize on-node communication
  - Example: nearest neighbor exchange in 2 dimensions
    - Sweep3d uses a 2-D grid for communication
- Determine whether or not a custom MPI rank order will produce a significant performance benefit
- Grid detection is helpful for programs with significant point-to-point communication
- Produce a custom rank order if it's beneficial based on grid size, grid order and cost metric

Automatic Grid Detection Example Summary

Example summary for sweep3d (pat_report table Notes)

This application appears to use point-to-point MPI communication at least partly organized into a 8 X 6 grid pattern. Time spent in MPI routines accounted for over 63.1% of the execution time. A portion of this time could potentially be saved by utilizing a rank order that maximizes the fraction of communication that is between ranks on the same node. The following table estimates this fraction for several rank orders.

An MPICH_RANK_ORDER file was generated along with this report and contains the Custom rank order from the following table. This file also contains usage instructions and a table of alternative rank orders.

Automatic Grid Detection Example Table

Table 4: Sent Message Stats for Selected MPI Rank Orders

Rank Order	On-Node Bytes/PE	On-Node Bytes/PE% of Total Bytes/PE	Options for grid_order utility
Custom	1.30e+07	50.00%	-R -P -m 48 -n 4 -g 8,6 -c 2,1
SMP	8.10e+06	31.25%	
Fold	6.75e+05	2.60%	
RoundRobin	0.00e+00	0.00%	

MPICH_RANK_ORDER File Example

```
# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:      /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File:     sweep3d.mpi+pat+27054-89t.ap2
# Number PEs:   48
# Max PEs/Node: 4
#
# To use this file, set the environment variable
# MPICH_RANK_REORDER_METHOD to 3 prior to executing the program.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.
…
```

Loop Statistics

- Helps identify loops to move to GPU:
  - Loop timings approximate how much work exists within a loop
  - Trip counts can be used to help carve up loop on GPU

- Enabled with CCE –h profile_generate option

- Loop statistics reported by default in pat_report table

Example Loop Stats

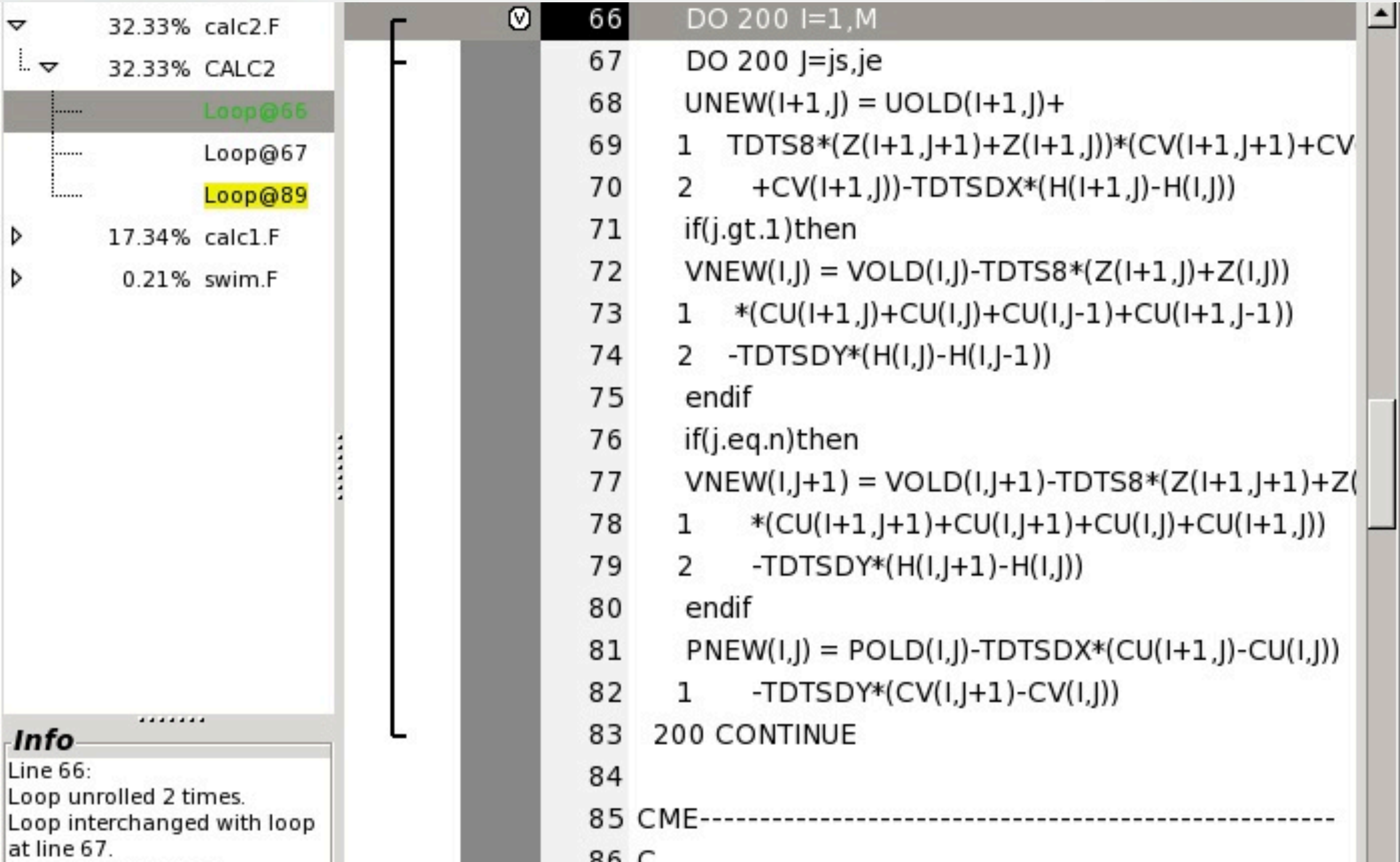
```
Notes for table 2:
  Table option:
    -O loops
  …
  The Function value for each data item is the avg of the PE values.
    (To specify different aggregations, see:  pat_help report options s1)

  This table shows only lines with Loop Incl Time / Total > 0.0095.
    (To set thresholds to zero, specify:  -T)

  Loop data version: L.12.2:B.3.1

  Loop instrumentation can interfere with optimizations, so time
  reported here may not reflect time in a fully optimized program.

  Loop stats can safely be used in the compiler directives:
   !PGO$       loop_info est_trips(Avg) min_trips(Min) max_trips(Max)
   #pragma pgo loop_info est_trips(Avg) min_trips(Min) max_trips(Max)

  Explanation of Loop Notes (P=1 is highest priority, P=0 is lowest):
   novec (P=0.5): Loop not vectorized (see compiler messages for reason).
   sunwind (P=1): Loop could be vectorized and unwound.
   vector (P=0.1): Already a vector loop.
```

Example Loop Stats (2)

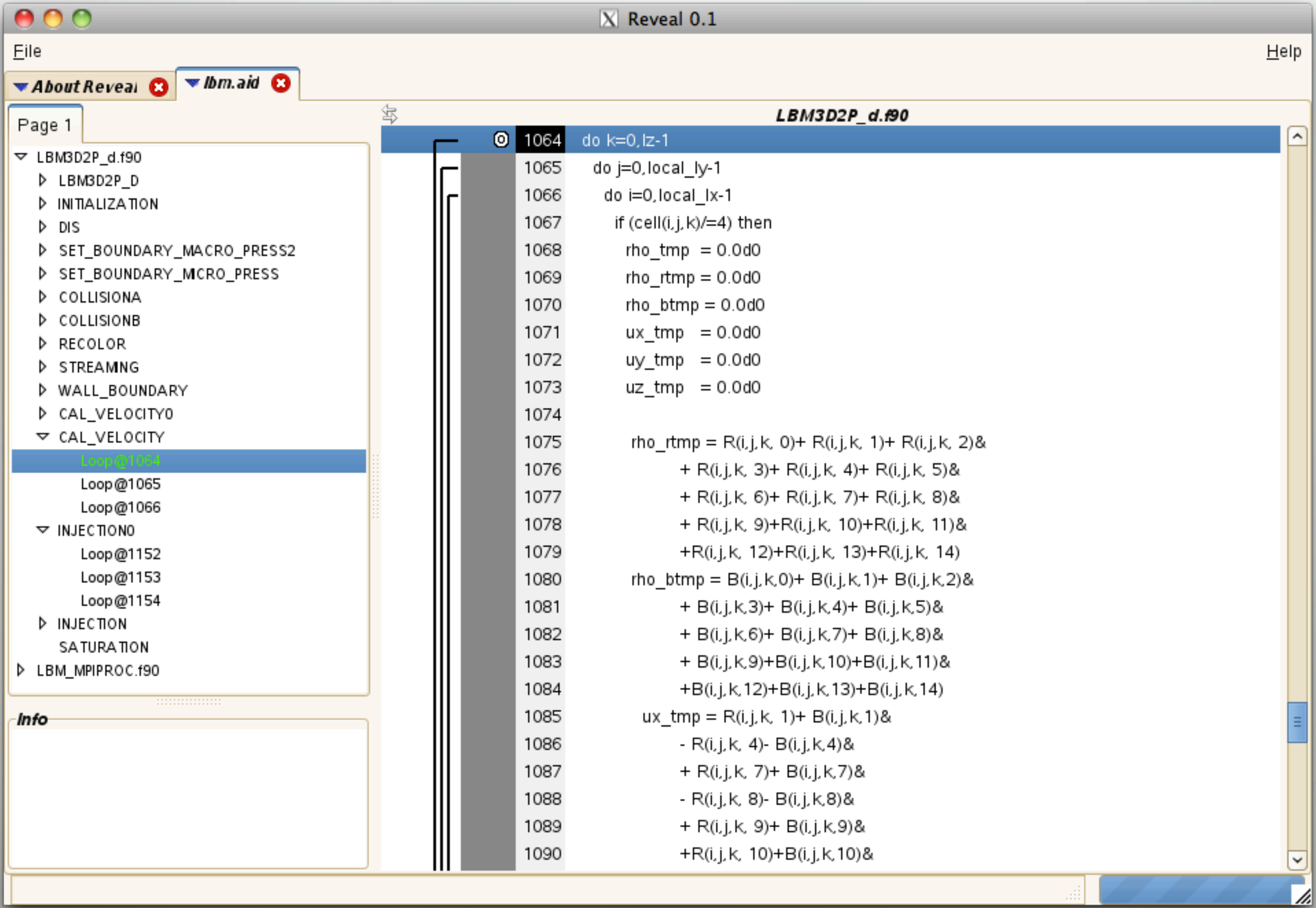
Table 2: Loop Stats from -hprofile_generate

Loop Incl Time / Total	Loop Incl Time	Loop Incl Time / Hit	Loop Hit	Loop Trips Avg	Loop Notes	Function=/.LOOP\. PE='HIDE'
24.6%	0.057045	0.000570	100	64.1	novec	calc2_.LOOP.0.li.614
24.0%	0.055725	0.000009	6413	512.0	vector	calc2_.LOOP.1.li.615
18.9%	0.043875	0.000439	100	64.1	novec	calc1_.LOOP.0.li.442
18.3%	0.042549	0.000007	6413	512.0	vector	calc1_.LOOP.1.li.443
17.1%	0.039822	0.000406	98	64.1	novec	calc3_.LOOP.0.li.787
16.7%	0.038883	0.000006	6284	512.0	vector	calc3_.LOOP.1.li.788
9.7%	0.022493	0.000230	98	512.0	vector	calc3_.LOOP.2.li.805
4.2%	0.009837	0.000098	100	512.0	vector	calc2_.LOOP.2.li.640

Source Code – Loopmark



Display Scoping Information for Selected Loop



Display Scoping Information for Selected Loop (2)

OpenMP Construct

LBM3D2P_d.f90: lines 1064 -> 1130

Name	Type	Scope	F	L	Info
b	Scalar	Shared			
cell	Scalar	Shared			
local_lx	Scalar	Shared			
local_ly	Scalar	Shared			
lz	Scalar	Shared			
r	Scalar	Shared			
rho	Scalar	Shared			
rho_btmp	Scalar	Private	N	N	
rho_rtmp	Scalar	Private	N	N	
rho_tmp	Scalar	Private	N	N	
ux_tmp	Scalar	Private	N	N	
uxyz	Scalar	Shared			
uz_tmp	Scalar	Private	N	N	

Example Performance Statistics

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

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

 100.0% | 18.113521 |       -- |     -- |    6.0 |Total
|-----------------------------------------------------------------
| 100.0% | 18.113443 |       -- |     -- |    5.0 |USER
||----------------------------------------------------------------
||  90.6% | 18.113000 | 0.000000 |   0.0% |    1.0 |acc_sample_.ACC_DATA_REGION@li.23
||   9.4% |  0.000443 | 0.000000 |   0.0% |    1.0 |acc_sample_.ACC_REGION@li.24
||================================================================
|   0.0% |  0.000078 | 0.000000 |   0.0% |    1.0 |ETC
||----------------------------------------------------------------
|   0.0% |  0.000078 | 0.000000 |   0.0% |    1.0 | exit
|=================================================================
```

Example Performance Statistics

```
Table 2:  Time and Bytes Transferred for Accelerator Regions

  Host   |    Host   |     Acc   | Acc Copy  | Acc Copy  | Calls   |Calltree
 Time%   |    Time   |    Time   |        In |       Out |         |
         |           |           | (MBytes)  | (MBytes)  |         |
 100.0% |   18.113 |   18.112 |   209.808 |   209.808 |       4 |Total
|-----------------------------------------------------------------------------------------
| 100.0% |   18.113 |   18.112 |   209.808 |   209.808 |       4 |acc_sample_
|        |          |          |           |           |         | acc_sample_.ACC_DATA_REGION@li.23
|||---------------------------------------------------------------------------------------
3||   90.6% |   16.418 |       --- |       --- |       --- |       1 |sync
3||    9.4% |    1.695 |     1.695 |   209.808 |   209.808 |       2 |transfer
3||    0.0% |    0.000 |    16.418 |     0.000 |     0.000 |       1 |acc_sample_.ACC_REGION@li.24
4||         |          |           |           |           |         | async_kernel
|=========================================================================================
```

The Next Generation of Debuggers on Cray Systems

- Systems with hundreds of thousands of threads of execution need a new debugging paradigm
  - Innovative techniques for productivity and scalability
    - Scalable Solutions based on MRNet from University of Wisconsin

      STAT - Stack Trace Analysis Tool
      - » Scalable generation of a single, merged, stack backtrace tree
        - running at 216K back-end processes

      ATP - Abnormal Termination Processing
      - » Scalable analysis of a sick application, delivering a STAT tree and a minimal, comprehensive, core file set.
    - Comparative debugging
      - A **data-centric paradigm** instead of the traditional control-centric paradigm
      - Collaboration with Monash University and University of Wisconsin for scalability
    - Fast Track Debugging
      - Debugging optimized applications
      - Added to Allinea's DDT 2.6 (June 2010)