Instrumentation + Sampling

BSC Performance Tools
- Events correlated to specific program activity
  - Start/exit iterations, functions, loops,…

- Different intervals:
  - May be very large, may be very short
  - Variable precision

- Captured data:
  - Hardware counters, call arguments, call path,…. 
Sampling

- Events uncorrelated to program activity (at least not specific)
  - Time (or counter) overflow

- Controlled granularity:
  - Sufficiently large to minimize overhead
  - Guaranteed acquisition interval/precision

- Statistical projection
  - %Counts = %time (or metric)
  - Assuming no correlation, sufficiently large #samples
Sampling

- An option in the Extrae xml file
  - Requires –enable-sampling at configure step

```xml
<counters enabled="yes">
  <cpu enabled="yes" starting-set-distribution="1">
    <set enabled="yes" domain="all" changeat-globalops="0">
      PM_CYC, PM_DATA_FROM_MEM, PM_GCT_FULL_CYC, PM_INST_CMPL, PM_INST_DISP, PM_LD_MISS_L1, PM_LD_REF_L1, PM_ST_REF_L1
      <sampling enabled="yes" frequency="100000000">PM_CYC</sampling>
    </set>
  </cpu>
</counters>
```

- Warning: Read counters on the user functions if they are going to be used as folding boundaries
- Standard hwc cfgs can be used on the generated trace
- Special cfgs for sampling call stack (in /gpfs/apps/CEPBATools/etc/cfgs/sampling+folding)
High frequency periodic sampling

- Sampling frequency > Nyquist
- Shows fine structure
- May have overhead

- Identification of function span
  - Assumption: consecutive samples with same call stack → whole interval assigned to function
  - All functions / specific subsets
    - Top of stack
    - Walk the stack searching for the first routine in the target set.
Overflow Sampling

- Possible to sample on any hwc overflow
- Useful to identify Internal behaviour i.e. Density of L1 misses within MPI in an SMP
  - overflows inside the MPI calls when data actually arrives (which generates a lot of L1 misses.)
Instrumentation + sampling

- Both

- Guaranteed interval
- Captured data:
  - Hardware counters (since previous probe)
  - call path
  - Call arguments in some probes
- How to use it?

- Can we give detail at the level of tenths of nanoseconds acquiring data at milisecond intervals?
- Instrumentation → Reference
  - Identify different instances of a region for which to obtain detailed time evolution of metrics
    - Stationary behaviour assumed
  - Target region:
    - Iteration
    - Routine
    - Routine excluding MPI calls
    - ...

```plaintext
fA
MPICall
fB
fA
MPICall
fB
...
```
New roles

- Sampling role $\rightarrow$ relative data
  - Guarantee granularity
  - Provide data to increase granularity
Folding counters: Projecting

- WARNING: requires stationary application behaviour
- Basic idea: Fold samples relative to reference. Compute cumulative count since reference

- Folding process has to:
  - take care of variance/noise in input data.
  - Model (fit) cumulative hardware counter count
  - Also fold sampled call stack
Folding results: trace with synthetic hwc events

- Result: trace for one iteration with synthetic paraver events
- Refer counts/timestamps to start of iteration
  - Call stack
    - Search for consecutive sequences of folded samples within same function and generate synthetic events
  - Hardware counters
    - Noise reduction
    - Fit folded samples
    - Sample fitting curve to generate synthetic events.
Folding results: gnuplot of hwc counter metrics

- Cumulative count since beginning of region (routine/cluster)
- Instantaneous performance metric (HW count/time, i.e. MFLOPS, ...)

**Task 0 Thread 0 - tree_domains,32**
Duration = 284.68 ms Counter = 76856.56 Kinstructions

**Task 0 Thread 0 - tree_build,0**
Duration = 1233.49 ms Counter = 138486.13 Kinstructions

Normalized MIPS @ Routines in PEPC
Folding result: reference to source code

Source line

Instantaneous MIPS

4 iterations outer loop

4 x_solve_cell

4 backsubstitute
Beware of correlations

- 50 Mcycles seemed a reasonable interval
- Ends up being slightly correlated to application periodicity 😊
  - Can bias a pure statistical profile
  - Folding has potential to reduce this effect
Comparing Marenostrum to SGI Altix

Marenostrum, NPB 3.2
BT benchmark
Class = B
Processes = 16

SGI Altix, NPB 3.2
BT benchmark
Class = B
Processes = 16
Examining COPY_FACES routine on Marenostrum

Task 4 Thread 0 -- COPY_FACES:{{copy_faces.f}{4,7}-{320,9}}

- Normalized time
- Samples
- Curve fitting
- Curve fitting slope
- Slope of PAP_L_T_INS
- PAP_L_T_INS

Task 4 Thread 0 -- COPY_FACES:{{copy_faces.f}{4,7}-{320,9}}

- Normalized time
- Samples
- Curve fitting
- Curve fitting slope
- Slope of PAP_L_SR_INS
- PAP_L_SR_INS

Task 4 Thread 0 -- COPY_FACES:{{copy_faces.f}{4,7}-{320,9}}

- Normalized time
- Samples
- Curve fitting
- Curve fitting slope
- Slope of PAP_L_DCM
- PAP_L_DCM

Task 4 Thread 0 -- COPY_FACES:{{copy_faces.f}{4,7}-{320,9}}

- Normalized time
- Samples
- Curve fitting
- Curve fitting slope
- Slope of PAP_L_D_INV
- PAP_L_D_INV
Sampling & Folding

- A set of utilities to process a trace
  - Extract: generates intermediate data from the trace
  - Interpolate: performs the data interpolation and generates output

- Inputs: the trace
  - The trace must have been obtained specifying sampling (in xml. See following slides)
    - Frequency of sampling can be very low (i.e. >10ms)
  - The trace must have reference events. Either:
    - User functions: using manual or automatic instrumentation. (See instrumentation session)
    - Cluster events: using a previously clusterized sampled trace.

- Outputs:
  - Gnuplots with instantaneous evolution of performance metrics
  - Trace will injected synthetic fine grain hardware counters
    - Special configuration files are needed to visualize the instantaneous evolution of the metrics with paraver. Some example configuration files are included in /gpfs/apps/CEPBATOOLS/etc/cfgs/sampling+folding.
    - The same cfgs previously described for just sampling can also be used on the folded trace.
Sampling & Folding steps

1. Decide which event you want to use as reference
   - User functions, iterations – need to include it on the tracefile
   - Clusters

2. Obtain a tracefile with sampling

3. If using clusters as reference, clusterize the obtained trace

4. Run the folding tools
   - preprocess.sh (Example: preprocess.sh bt.B.16 user_functions)
   - interpolate.sh (Example: interpolate.sh bt.B.16 0.0 -counter PAPI_TOT_INS -counter PAPI_BR_INS)
   - Analyze the results – scatter plot, *folded paraver tracefile*

- Files available in PRACE_Autumn_School/examples/sampling
#!/bin/bash

FOLDING_BIN=/gpfs/apps/CEPBATOOLS/folding

tracename=$1
shift

if [[ "$2" == "sort_callers" ]]; then
    $FOLDING_BIN/sort_pcf $tracename.prv
    trace=$tracename.sorted.prv
else
    trace=$tracename.prv
fi

if [[ "$1" == "clusters" ]]; then
    $FOLDING_BIN/fuse $tracename.prv
    trace=$tracename.fused.prv
    $FOLDING_BIN/extract -separator 90000001 $trace
    elif [[ "$1" == "user_functions" ]]; then
    $FOLDING_BIN/extract -separator 60000019 -phase-separator 50000001 -phase-separator 50000002 -phase-separator 50000003 $trace
else
    echo "Usage: folding.sh <tracefile> [user_functions|clusters] [sort_callers]"
fi

For more information see /gpfs/apps/CEPBATOOLS/folding/README
Internals of interpolate.sh

#!/bin/bash

FOLDING_BIN=/gpfs/apps/CEPBATOOLS/folding

trace=$1
shift
thread=$1
shift
counters_option=$*

# -remove-outliers 2.0 : removes outliers beyond the interval of confidence of the 95%
# -min-duration 1      : do not emit information for bursts shorter than 1ms

$FOLDING_BIN/interpolate -remove-outliers 2.0 -min-duration 1 $counters_option -generate-gnuplot yes $trace.extract.$thread

For more information see /gpfs/apps/CEPBATOOLS/folding/README
• Extract data from the tracefile. This will depend on the data you want to fold.

• Example 1: **Folding user functions**
  
  extract -separator 60000019 tracefile.prv

• Example 2: **Folding user functions and separating its computation regions using MPI calls**
  
  extract -separator 60000019 -phase-separator 50000001 -phase-separator 50000002 -phase-separator 50000003 tracefile.prv

• Example 3: **Folding clusters**
  
  extract -separator 90000001 tracefile.prv

• Any of these commands will report by task & thread extracted files with folded sampled data. The files will look like tracefile.extract_TASK.THREAD, where TASK and THREAD are values from 0 to MAXTASK-1 or 0 to MAXTHREAD-1 respectively.
Clustering steps – interpolate

• Interpolate extracted data.

  Example 1: Interpolate total instructions and generating gnuplots of task 0 thread 3.
  
  interpolate -counter PAPI_TOT_INS -generate-gnuplot yes tracefile.extract.0.3

  Example 2: Interpolate total instructions and l2D cache misses and generating gnuplots of task 3 thread 1.
  
  interpolate -counter PAPI_TOT_INS -counter PAPI_L2_DCM -generate-gnuplot yes tracefile.extract.3.1

  Example 3: Interpolate total instructions and l2D cache misses and generating gnuplots of task 3 thread 1, now removing outliers that are outside the 95% of Interval of Confidence -using time & instructions -. 
  
  interpolate -counter PAPI_TOT_INS -counter PAPI_L2_DCM -generate-gnuplot yes -remove-outliers 2.0 tracefile.extract.3.1
Clustering steps – interpolate (and 2)

- **Example 4:** Interpolate total instructions and L2D cache misses and generating gnuplots of task 3 thread 1 removing computation phases that last less than 1 millisecond.
  
  ```
  interpolate -counter PAPI_TOT_INS -counter PAPI_L2_DCM -generate-gnuplot yes -min-duration 1 tracefile.extract.3.1
  ```

- **Example 5:** Interpolate total instructions of task 0 thread 0 and reintroduce on the Paraver tracefile using the event type/value marker 123456/1
  
  ```
  interpolate -counter PAPI_TOT_INS -feed-region 123456 1 tracefile.extract.0.0
  ```

- **Example 6:** Interpolate total instructions of task 0 thread 0 and reintroduce on the Paraver tracefile between timestamps 100 and 200
  
  ```
  interpolate -counter PAPI_TOT_INS -feed-time 100 200 tracefile.extract.0.0
  ```