

A Proposal for a Profiling Data Exchange Format

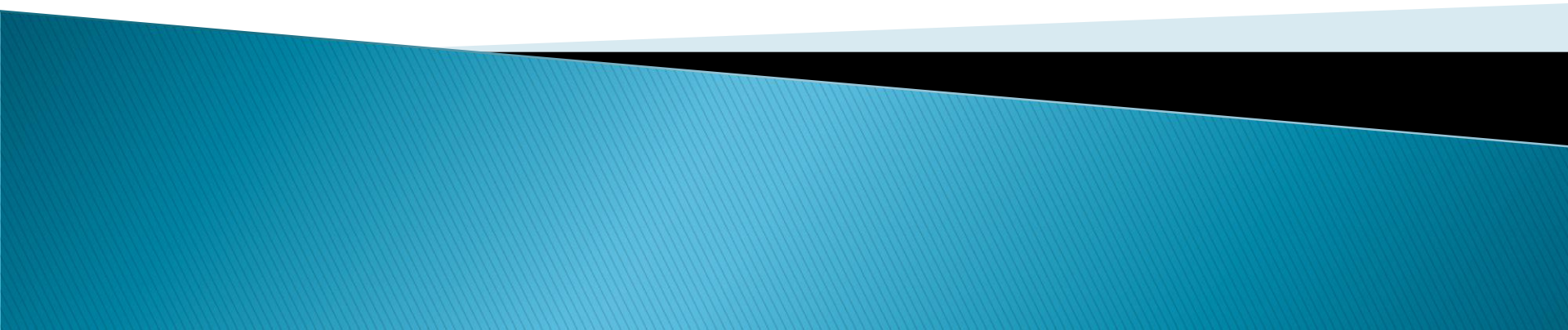
Bernd Mohr, JSC

Martin Schulz, LLNL

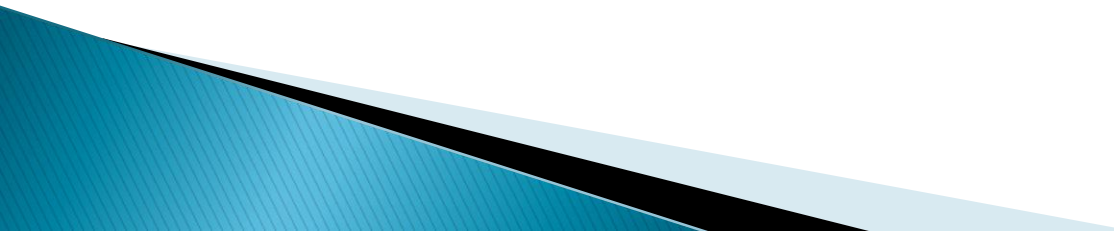
Dan Gunter, LBL

Kevin Huck/Wyatt Spear, University of Oregon

Xingfu Wu, Texas A&M



Goals & Purpose

- ▶ Data exchange between tools
 - Create data in one tool
 - Display in another
 - ▶ Tool independent long term storage
 - Maintain all relevant information in one file
 - Storage in performance databases
 - ▶ Interface for PERI auto tuning efforts
- 

History

- ▶ Started as PERI–XML effort
 - Format to describe meta data
 - Used to track experiments in a single database
- ▶ Extensions to data were seen as useful
 - First attempts demonstrated at SC 2007
 - Missing features and generality
 - More general discussion at CScADS 2008
 - Decision to move forward with broader participation
- ▶ Regular phone meetings since then
 - Core team: JSC, LBL, LLNL, TAMU, UO
 - Today: first complete draft specification

Approach

- ▶ Performance data as five dimensional space, each represented as one or more hierarchies
 - **<metrics>**
 - **<code>**
 - **<binaries>**, **<sources>**, **<programs>**
 - **<space>**
 - **<machines>**, **<applications>**, **<mappings>**, **<topologies>**
 - **<time>**
 - **<state>**
- ▶ Keep PERI-XML meta data (**<runrules>**)

Requirements / Guidelines

- ▶ Many entries are optional
 - Don't force tools to provide data they don not have
- ▶ Unknown xml elements are ignored
 - Allows tools to represent important tool-specific features not covered by PERI-XML

<metrics> Dimension

- ▶ Describe what is measured and in which unit
 - Can be organized as a hierarchy
- ▶ Derived metrics
 - Allows simple combination of metrics
 - Operators: +, -, *, /
- ▶ Aggregation metrics
 - Combine data across one or more dimensions
 - Operators: min, max, sum, sample

<metrics> Dimension (XML)

```
<metrics>
  <metric id="hw0" units="none" type="int" name="PAPI_FP_INS">
    <desc>Number of floating point instructions</desc>
  </metric>

  <metric id="mp0" units="bytes" type="int" name="Bytes transferred">
    <metric id="mp1" units="bytes" type="int" name="Bytes send" />
    <metric id="mp2" units="bytes" type="int" name="Bytes recv" />
    <derived>
      <metric ref="mp1" />
      <metric ref="mp2" />
      <op name="+" />
    </derived>
  </metric>
</metrics>
```

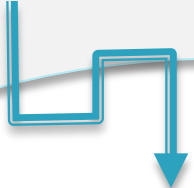
<code> Dimension

- ▶ Encodes the program(s) involved
 - <sources> and <binaries> to specify files involved
 - Using hierarchy to describe inclusion
 - <programs> defines static and dynamic structure
 - Using hierarchy to describe nesting
 - <regions> identifies static code regions of interest
 - Functions, loops, statements, basic blocks, phases, ...
 - Map regions to source files
 - <cnodes> describes dynamic region tree

<code> Dimension (XML)

```
<code>
  <binaries>
    <binary id="exe0"
            type="executable"
            arch="x86_64">
      <file>foo</file>
    </binary>
  </binaries>

  <sources>
    <source id="src0" type="code">
      <name>foo.c</name>
    </source>
  </sources>
```



```
01: void foo(int i) { /* ... */ }
02:
03: int main() {
04:   foo(23);
05:   foo(42);
06: }
```

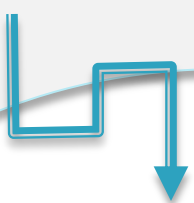
```
<program id="prg0">
  <regions>
    <region id="r0" type="routine">
      <name>main</name>
      <src ref="src0">
        <line first="3" last="6" />
      </src>
      <bin ref="exe0" />
    </region>
    <region id="r1" type="routine">
      <name>foo</name>
    </region>
  </regions>
  <cnodes>
    <cnode id="c1" ref="r0">
      <cnode id="c1.1" ref="r1" />
      <cnode id="c1.2" ref="r1" />
    </cnode>
  </cnodes>
</program>
</code>
```

<space> Dimension

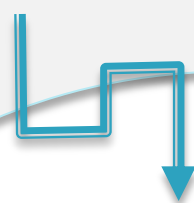
- ▶ Describes structure of machines and applications under investigation
 - <machines> describes hardware execution platform
 - Hierarchy of <node>, <cpu>, <core>
 - Additional optional levels with <unit type="XXX">
 - Organize machines with <group> to e.g. grids or sites
 - <applications> specifies software structure
 - PID and TID data via <process> and <thread>
- ▶ Optional
 - <mappings> information of applications to machines
 - <topologies> HW and SW topology information
 - E.g. for communicators, thread groups, MPI topologies, ...

<space> Dimension (XML)

```
<space>
<machines>
  <machine id="ma0" arch="pwr6"
           vendor="IBM"
           vendorid="p6 575">
    <name>JUMP</name>
    <unit id="r42" type="rack">
      <name>Rack 42</name>
      <node id="node13">
        <name>jump01</name>
      </node>
      <node id="node14">
        <name>jump02</name>
      </node>
    </unit>
  </machine>
</machines>
```



```
<applications>
  <application id="app0"
               exeref="exe0">
    <name>foo</name>
    <process id="p0" pid="1234">
      <thread id="t0.0" tid="0" />
    </process>
    <process id="p1" pid="5783">
      <thread id="t1.0" tid="0" />
    </process>
  </application>
</applications>
```



<space>Dimension (XML) II

```
<mappings>
  <mapping id="map0">
    <runson aref="p0" mref="node13" />
    <runson aref="p1" mref="node14" />
  </mapping>
</mappings>

<topologies>
  <cartesian id="com0" type="communicator" ndims="1">
    <name>MPI_COMM_WORLD</name>
    <dim size="2" periodic="false" />
    <coord ref="p0">0</coord>
    <coord ref="p1">1</coord>
  </cartesian>
</topologies>
</space>
```

<space> Dimension (Shortcuts)

- ▶ For describing very large regular machines
- ▶ Similar syntax for <applications>, <mappings>, and <cartesian>

```
<machine id="ma1" arch="ppc450"
        vendor="IBM"
        vendorid="BG/P">
  <name>JUGENE</name>
  <unit_seq type="rack"
            first="1" last="72">
    <node_seq num="1024">
      <cpu_seq num="1">
        <core_seq num="4" />
      </cpu_seq>
    </node_seq>
  </unit_seq>
</machine>
```

- ▶ This automatically defines id's for
 - (unit) ma1:1 to ma1:72
 - (nodes) ma1:1.0 to ma1:72.1023
 - (cpus) ma1:1.0.0 to ma1:72.1023.0
 - (cores) ma1:1.0.0.0 to ma1:72.1023.0.3

<time> Dimension

- ▶ Hierarchy of time intervals
 - Measured in absolute time since program start
- ▶ Note: for code delimited intervals use "phases"

```
<time>  
  <interval id="time0" units="ms">  
    <name>initialization</name>  
    <start>500333</start>  
    <end>600444</end>  
  </interval>  
</time>
```

<states> Dimension

- ▶ Extra dimension for further qualification of the measured data, e.g., parameter profiling
 - Currently syntax for specifying parameter/value pairs

```
<states>  
  <state id="msg4">  
    <p nm="message size">4</p>  
  </state>  
  <state id="msg8">  
    <p nm="message size">8</p>  
  </state>  
</states>
```

<data> Section

- ▶ Each data point provides a value for a vector
<*code, space, time, state, metrics*>
- ▶ Hierarchical description of data
 - For minimal representation
 - Inherit parts of vector from higher level
 - Order of dimensions left to the producer
 - To allow efficient generation
 - Might need more efficient (outside of XML) representation for very large measurements
- ▶ Reference to non-leaf hierarchy identifiers allows to store precalculated summary metrics

<data> (XML)

```
<data>
  <d metric="mp0">
    <d state="msg4">
      <d code="r0">
        <d spc="p0">32872980</d>
        <d spc="p1">45960582</d>
      </d>
      <d code="r1">
        <d spc="p0">509386</d>
        <d spc="p1">3029745</d>
      </d>
    </d>
  </d>

  <!-- ... -->

</d>
</data>
```

Tool Testcases

- ▶ KOJAK/Scalasca (www.scalasca.org)
 - CUBE callpath and trace-pattern profiles
- ▶ IPM (ipm-hpc.sourceforge.net)
 - Both summary and detailed profiling modes
- ▶ Open|SpeedShop (www.openspeedshop.org)
 - O|SS preprocesses data before exporting
 - Stresses aggregation metrics features
- ▶ TAU (tau.uoregon.edu)
 - Flat, call path, parameter ... profiles
- ▶ Prophecy (prophecy.cs.tamu.edu)
- ▶ CrayPat (docs.cray.com/books/S-2376-41/)
 - ▶ TODO: gprof, mpiP, ompP, hpctoolkit, peekperf

Open Issues

- ▶ How to represent inclusive/exclusive time?
 - Independent two metrics or ???
- ▶ Scalable non-XML(?) binary(?) data representation
- ▶ More elaborate **<state>** and **<time>** specifications
- ▶ Complete current spec and further info (work in progress!!!)
 - http://www.peri-scidac.org/wiki/index.php/PERI_XML

Feedback wanted ...

- ▶ Can all tools map to this multi-dimensional performance data space?
 - ▶ Does this format cover the full spectrum of data exported by tools?
 - ▶ How easy/difficult is this format to work with for performance databases?
 - ▶ How complex are readers?
- 