Performance Analysis Objective

How is my application performing?

Can I describe it in a simple way? Quantitatively?

Is there anything I can do to improve its performance?

What?

Specific

Expected performance improvement

Preferably with minimum effort/cost
Who can I blame?
CEPBA – tools framework

Trace handling & display

Simulators

Open Source
Linux and windows

XML control

Valgrind
OMPITrace
MRNET
Dyninst, PAPI

.trf

Time Analysis, filters

.DIMEMAS
VENUS (IBM-ZRL)

Instr. Level
Simulators

Machine description

Paraver

.PeekPerf

Stats Gen

XML
control

.Open Source
Linux and windows

Ctrl. Level
Simulators

How2gen.xml

Data Display Tools

Valgrind
OMPTTrace
MRNET
Dyninst, PAPI

Time Analysis, filters

.DIMEMAS
VENUS (IBM-ZRL)

Instr. Level
Simulators

Machine description

Paraver

.PeekPerf

Stats Gen

XML
control
Extrae - Instrumentation
Parallel Program Instrumentation: Features

- **Probe injection mechanisms**
  - Library Preload: Linux clusters (Dynamically linked MPI library)
  - Static linking: BG
  - Compiler-based instrumentation (PDT) ALTIX, PowerPC
  - Dynamic instrumentation (Dyninst): ALTIX, PowerPC (beta version)

- **Captured events:**
  - MPI calls (including I/O)
  - Hardware counters: PAPI: standard + native
    - Several sets: Rotate groups periodically / Different groups per processes
  - Network: GM at end of trace, MX at flushes
  - OS counters: At the end of the trace and when flushing
  - Link to source:
    - Dyninst based systems: user function events on entry and exit (for selected functions).
    - Library preload: MPI caller (several levels)
  - User events (API)

- **Towards a unified tracing package and specification**
  - xml control file
A typical MareNostrum process

```bash
#!/bin/bash

# @ total_tasks = 2
# @ cpus_per_task = 1
# @ tasks_per_node = 2
...
srun ./trace.sh ./mpi_ping
```

```bash
export#!/bin/sh

export MPITRACE_HOME=/gpfs/apps/CEPBATOOLS/mpitrace/64
export MPTRACE_CONFIG_FILE=mpitrace.xml

export LD_PRELOAD=${MPITRACE_HOME}/lib/libmpitrace.so

## Run the desired program
$*
```

Examples of MN set-up available at /gpfs/apps/CEPBATOOLS/tracing-setup
Paraver - Visualization and analysis
New version of Paraver

- GUI based on wxWidgets – Linux and Windows
Views: Timelines

- From raw events $\rightarrow$ Piece-wise constant functions of time $\rightarrow$ plots / colors

- Basic metrics

- Derived metrics

- Models

\[
\text{useful \_IPC} = \frac{\#\text{instr} \ast \text{useful}}{\#\text{cycles}}
\]

\[
\text{MPI \_call \_Cost} = \frac{\text{MPI \_call \_duration}}{\#\text{bytes}}
\]

\[
\text{preempted \_time} = \text{elapsed} - \frac{\text{cycles}}{\text{clock \_freq}}
\]

\[
\text{L2 \_miss \_latency} = \frac{\#\text{cycles} - \#\text{instr} / \text{idealIPC}}{\#\text{L2misses}}
\]
Views: Statistics

- 2D
  - Profiles
  - Histograms
  - Correlations
  - Communication Patterns

- 3D
  - Duration - IPC
    - While communicating
    - While computing
Scalability
Scalability of Presentation

- Aggregation
  - Functional rather than scalability motivation

- Display
  - Non linear render
    - Value for pixel
    - Colors

- Objects
  - Any subset

NAS LU-MZ. MPI+OpenMP. 121 procs @ SP Nighthawk
Scalability of Presentation

- Linpack @ Marenostrum: 10k cores x 1700 s

<table>
<thead>
<tr>
<th></th>
<th>Duration</th>
<th>Linpack</th>
<th>Dgemm duration</th>
<th>11.8 s</th>
<th>10 s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dgemm</td>
<td>IPC</td>
<td>L1 miss ratio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.95</td>
<td>2.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>0.7</td>
</tr>
</tbody>
</table>
Scalability: Data reduction through periodicity analysis

- Data handling/summarization capability
  - Software counters, filtering and cutting
  - Supported by GUI.

- Automatizable through signal processing techniques:
  - Mathematical morphology to clean up perturbed regions
  - Wavelet transform to identify coarse regions
  - Spectral analysis for detailed periodic pattern

- Algorithms currently applied to traces
  - to be ported to run time

- Similar to manual structure identification
Scaling by selectively emitting trace data

- Gadget: 15.5 MB (3 MB compressed) → Init, 3 iters (10.6 s); termination

2048 procs

Useful duration

% MPI time

# collectives

Collective bytes

# p2p

p2p bytes

p2p BW
Dimemas: Models and predictions
CEPBA-Tools Environment

Predictions/expectations

CEPBA-Tools training. PRACE Autumn School. 2010
Dimemas: Coarse grain, Trace driven simulation

- Simulation: Highly non linear model
  - Linear components
    - Point to point communication
    - Sequential processor performance
      - Global CPU speed
      - Per block/subroutine
  - Non linear components
    - Synchronization semantics
      - Blocking receives
      - Rendezvous
    - Resource contention
      - CPU
      - Communication subsystem
        - links (half/full duplex), busses

\[ T = \frac{\text{MessageSize}}{\text{BW}} + L \]
Example: Specfem3D

- Should I introduce asynchronous communication?

![Image of Specfem3D simulation](image)

![Image of MPI call in Specfem3D](image)

Real
ideal
NM prediction
Prediction 100MB/s
Prediction 10MB/s
Prediction 5MB/s
Prediction 1MB/s

Courtesy Dimitri Komatitsch
Example: Network sensitivity

- Simulations with 4 processes per node
- NMM Iberia 4Km
  - Not sensitive to Latency
  - 512 sensitive to contention?
  - 256 MB/s OK
- ARW Iberia 4 Km
  - Not sensitive to Latency
  - sensitive to contention
  - Need 1GB/s

Impact of latency (BW=256; B=0)

Impact of BW (L=8; B=0)

Efficiency

Contention Impact (L=8; BW=256)
Clustering – Detecting structure
Clustering

- Useful for
  - Identifying and highlighting structure
  - Cluster information injected in trace
  - Phases within routines
  - Different routines may have similar behavior
- Compact trace encoding
- Input to time analysis
Clustering

- Useful for
  - Precise projection of hardware counters
  - Statistics
  - CPI stack model

<table>
<thead>
<tr>
<th>Cluster</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>%Time</td>
<td>54.88</td>
<td>17.96</td>
<td>16.90</td>
<td>6.44</td>
<td>1.42</td>
</tr>
<tr>
<td>Avg. Burst Dur. (ms)</td>
<td>1.02</td>
<td>0.78</td>
<td>13.14</td>
<td>2.50</td>
<td>1.11</td>
</tr>
<tr>
<td>IPC</td>
<td>1.02</td>
<td>0.65</td>
<td>0.89</td>
<td>0.91</td>
<td>6.53</td>
</tr>
<tr>
<td>MIPS</td>
<td>2231.8</td>
<td>1423.3</td>
<td>1966.5</td>
<td>2001.8</td>
<td>1163.0</td>
</tr>
<tr>
<td>MFLOPS</td>
<td>339.2</td>
<td>46.3</td>
<td>191.6</td>
<td>269.2</td>
<td>23.6</td>
</tr>
<tr>
<td>L1M/KInstr</td>
<td>0.92</td>
<td>1.53</td>
<td>1.19</td>
<td>1.17</td>
<td>2.88</td>
</tr>
<tr>
<td>L2M/KInstr</td>
<td>0.06</td>
<td>1.26</td>
<td>0.06</td>
<td>0.35</td>
<td>0.21</td>
</tr>
<tr>
<td>Mem.BW (MB/s)</td>
<td>16.79</td>
<td>218.47</td>
<td>13.87</td>
<td>85.77</td>
<td>29.76</td>
</tr>
</tbody>
</table>
MRNet integration: Scalable online analysis

Analysis at minute 30

Clusters distribution

CPI STACK model (generic)
Sampling – Adding details
Sampling

- Adding sampling information on the tracefile
  - based on the overflow mechanism offered by PAPI
    - period = f (cycles / instructions / cache misses...)
  - sampled information:
    - call stack – as reference to source
    - other hardware counters (not sampled)

- Useful to identify
  - Timeline profile
  - Application structure

Safe sampled functions

MFLOPS at each sampled interval

Instrumented MPI calls
Sampling

- Low sampling frequency (<< Nyquist)
  - Folding mechanism to achieve high precision
  - Assumes stationary periodic behavior
    - event at start of iteration

Detailed fine grain instructions within one iteration