KRASH
CPU Load Generation on Many-Core Machines

Guillaume Huard and Swann Perarnau

Moaïs INRIA Team, CNRS LIG Lab, Grenoble University
France

Tuesday, 20 April 2010
Outline

1. Motivations
2. Basic Notions
3. Existing CPU Load Generation Methods: an Overview
4. KRASH
5. Validation
6. Conclusion
Motivations

Evolution of computing machines

Towards Many-Core Systems

No more CPU frequency scaling.
Who has the biggest number of cores?

Simple system used as computing servers

- Single machine with 4-32 cores
- No use as a desktop
- Used as shared computing servers
Towards Heterogeneous Systems

Shared machines
- Multiple users, multiple programs in a single address space
- Fair Scheduling Policy

Specialized cores
- GPU inside CPU
- Variable frequencies
- Hybrid architectures (ex: Cell)
Shared Memory, Many-Core systems present a dynamic heterogeneity.
Motivations

Classical Parallel Programming Challenged

New Paradigms for Heterogenous Systems

- fine grained parallelization + work-stealing [Blumofe 1995]
- adaptive parallel algorithms [Roch & al. 2008]

New Evaluation Criteria

- Struggling capacity for resources access (out of our scope)
- Efficient use of given resources

How to Compare Them?

We must use a controlled environment.
Our Goal: Help experimental evaluation

Classical scientific methodology

- Identify a representative environment,
- Compare algorithms in this same experimental conditions.

Challenge: design a load generation tool

- Producing heterogeneous environment from dedicated machines.
- Guarantying reproducibility.
- Insensitive to other applications efforts to take control of resources.

In this first work, we focus on CPU load only.
Outline

1 Motivations

2 Basic Notions

3 Existing CPU Load Generation Methods: an Overview

4 KRASH

5 Validation

6 Conclusion
Basic Notions

Timeslicing and Resolution

Timeslice
Period of exclusive access to a CPU core.

CPU Load (during time interval)
Proportion of unavailability: core assigned to some process or in the kernel.

Timeslice Size
Quite large: typically 1ms to 10ms
  - Fair share between processes is only reached asymptotically.
  - Notion of scheduler resolution.
Example Load Profile

Load profile of a dynamic environment
(several NAS instances started and stopped at various times) run in concurrence
with a Linpack instance
A Good Load Generator

Requirements
- Precision, Reactivity
- Noise insensitivity

Must Avoid
- Intrusiveness
- Obtrusiveness
Outline

1 Motivations
2 Basic Notions
3 Existing CPU Load Generation Methods: an Overview
4 KRASH
5 Validation
6 Conclusion
Various methods

Run a CPU intensive process.  ⇒  No control on the load.
Various methods

Run a CPU intensive process. ⇒ No control on the load.

Adjust priority during execution. ⇒ Load sensitive to environment.
Various methods

Run a CPU intensive process. ⇒ No control on the load.

Adjust priority during execution. ⇒ Load sensitive to environment.

Realtime process. ⇒ Changes scheduler behavior.
### Various methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run a CPU intensive process.</td>
<td>No control on the load.</td>
</tr>
<tr>
<td>Adjust priority during execution.</td>
<td>Load sensitive to environment.</td>
</tr>
<tr>
<td>Realtime process.</td>
<td>Changes scheduler behavior.</td>
</tr>
</tbody>
</table>
Various methods

Run a CPU intensive process. ⇒ No control on the load.

Adjust priority during execution. ⇒ Load sensitive to environment.

Realtime process. ⇒ Changes scheduler behavior.


Scale CPU frequency. ⇒ Modifies hardware characteristics.
Existing Solutions Issues

50% load during NAS DT (communications intensive MPI application).

<table>
<thead>
<tr>
<th>Issue</th>
<th>Execution time</th>
<th>Slowdown</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>2.9</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>Real time priority</td>
<td>11.3</td>
<td>3.2</td>
<td>3.9 less comm./comp. overlapping</td>
</tr>
<tr>
<td>Start/Stop</td>
<td>NA</td>
<td>NA</td>
<td>&gt; 100 Supervision overhead</td>
</tr>
<tr>
<td>Frequency scaling</td>
<td>4.4</td>
<td>0.6</td>
<td>1.5 more comm./comp. overlapping</td>
</tr>
</tbody>
</table>

Current methods fail to load complex applications.
Outline

1. Motivations
2. Basic Notions
3. Existing CPU Load Generation Methods: an Overview
4. KRASH
5. Validation
6. Conclusion
Our claim

Interact directly with the Scheduler

- Precision, Reactivity ensured.
- Noise insensitive.
- (Very) Low Intrusiveness.
- No Obtrusiveness.
Our Load Generation Method

1. Use **cpu affinity** to attach a process (**boulder**) to each core.
2. Use **group scheduling** to assign timeslices to boulders.
3. use a **supervisor** process to adjust group priorities when needed.
C/C++ Implementation

- Linux only,
- Supervising process mostly sleeping,
- Using control cgroups VFS.

Input

```plaintext
cpu {
  ...  # begin cpuinj
  # config params
  profile {
    # the load profile itself
    0 {
      # cpuid to load
      0 70  # load 70% of the cpu
      60 30  # after one minute only 30% the cpu
      120 50
    }
    1 {  # another cpu loaded
      0 70
      10 30
    }
  }
  # end of profile
}
# end cpuinj
kill 150  # stop krash after 150 seconds
```

```
KRASH Load Reproduction

Reproduction performed by KRASH run in concurrence with a Linpack instance.
Outline

1 Motivations
2 Basic Notions
3 Existing CPU Load Generation Methods: an Overview
4 KRASH
5 Validation
6 Conclusion
In the Paper

Qualitative Comparison

- Dynamic load profile,
- Modification of the scheduler,
- Maximum number of loaded process,
- Resolution

Quantitative Comparison

- Precision and Intrusiveness
- Effects on I/O
- Effects on Network
- Effects on complex applications
In the Paper

Qualitative Comparison

- Dynamic load profile,
- Modification of the scheduler,
- Maximum number of loaded processes,
- Resolution

Quantitative Comparison

- Precision and Intrusiveness
- Effects on I/O
- Effects on Network
- Effects on complex applications
Quantitative comparison

KRASH is the only tool able to generate dynamic load profiles
⇒ comparison will be limited to constant load generation

Test platform: NUMA SMP machine
- 8 dual-core Opteron 875
- 32 GB RAM
- 250 GB Raid 1 storage subsystem
Obtrusiveness: Mixed (tasks duration, I/Os, CPU)

Constant 50% load during parallel gcc compilation (varying processes duration, mixed I/Os and CPU intensive tasks)

<table>
<thead>
<tr>
<th>Issue</th>
<th>Execution time</th>
<th>Slowdown</th>
<th>Issue</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>average</td>
<td>standard deviation</td>
<td></td>
</tr>
<tr>
<td>None</td>
<td>197</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>KRASH</td>
<td>387</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>Wrekavoc</td>
<td>NA</td>
<td>NA</td>
<td>&gt; 100</td>
</tr>
<tr>
<td>Real time priority</td>
<td>558</td>
<td>5</td>
<td>2.8</td>
</tr>
<tr>
<td>Frequency scaling</td>
<td>392</td>
<td>21</td>
<td>2</td>
</tr>
</tbody>
</table>

I/Os less important than in case I, per process overhead critical

⇒ Supervision overhead

⇒ RT FIFO prevents I/Os priority

⇒ None

⇒ None

⇒ Too many processes

⇒ None

⇒ None

⇒ None
Outline

1 Motivations
2 Basic Notions
3 Existing CPU Load Generation Methods: an Overview
4 KRASH
5 Validation
6 Conclusion
Conclusion and Future Works

Krash: a new CPU load generation method

- Behaves as would do a CPU intensive application,
- Reproducible load: precise, reactive and insensitive to noise,
- Unobtrusive: do not induces unexpected performance impact on other system resources,

Future Works: extend Krash

- To cache (application cache trashing),
- To memory (bandwidth sharing),
- To I/Os (bandwidth/latency perturbations).
Thank you for your attention.

Krash is publicly available at http://krash.ligforge.imag.fr/