KRASH: Reproducible CPU Load Generation on Many Cores Machines

Swann Perarnau and Guillaume Huard

INRIA Moais research team, CNRS LIG laboratory, Grenoble Universities, France
firstname.lastname@imag.fr

Abstract

In this article we present KRASH, a tool for reproducible generation of system-level CPU load. This tool is intended for use in shared memory machines equipped with multiple CPU cores which are usually exploited concurrently by several users. The objective of KRASH is to enable parallel application developers to validate their resources use strategies on a partially loaded machine by *replaying* an observed load in concurrence with their application. To reach this objective, we present a method for CPU load generation which behaves as realistically as possible: the resulting load is similar to the load that would be produced by concurrent processes run by other users. Nevertheless, contrary to a simple run of a CPU-intensive application, KRASH is not sensitive to system scheduling decisions. The main benefit brought by KRASH is this reproducibility: no matter how many processes are present in the system the load generated by our tool strictly respects a given load profile. To our knowledge, KRASH is the only tool that implements the generation of a dynamic load profile (a load varying with time). When used to generate a constant load, KRASH result is among the most realistic ones. Furthermore, KRASH provides more flexibility than other tools.

Categories and Subject Descriptors D.4.9 [*Operating Systems*]: Systems Programs and Utilities

General Terms Experimentation, Performance

Keywords CPU load generation, many cores, experimentation testbed

1. Introduction

New hardware architectures, composed of multiple cores and a shared memory, spread in the domain of High Performance Computing. These many-cores machines are generally used as computing servers and shared among several users. In this context, available resources to a given user are only a part of the shared machine: other processes constantly start and terminate and the operating system tries to fairly dispatch resources among them. At the end of the day, programs are faced with heterogeneous resource availability. To cope with this dynamic heterogeneity, new parallel programming paradigms have been put forward (work-stealing and adaptive parallel algorithms for instance). When comparing these different approaches, one might consider various evaluation criteria such as how they compete in the system to get resources control or how efficiently they use the resources they are given. In this article we do not address the evaluation of a parallel application capabilities to compete for resources access. The work we present is an experimental testbed able to control resources unavailability (what we name load) according to a determined pattern. Consequently, this work makes possible to compare the way distinct parallel programming paradigms use a fixed set of heterogeneous resources. This efficiency comparison is usually difficult as most system schedulers try to fairly balance resources access among running processes. In this context, a fine grained parallel application that spawns more processes than actual computing resources will get more access opportunities than other parallelization schemes, making the efficiency comparison impossible.

Thus, we propose a method to precisely produce and reproduce a controlled dynamic environment on a real dedicated machine. Our tool, named KRASH[1], is able to generate the same desired concurrent load, whatever the parallel program under evaluation, no matter how many processes it creates or the system resources it requires. In this article we focus on CPU load only, which is usually the most important factor in high performance computing.

2. Reproducible generation of CPU load

The principle is quite simple: place on each CPU core a CPU intensive process that will act as load generator. Then directly ask the scheduler to assign to our load generator the desired proportion of available timeslices. With our method, the generation is precise, because it is performed at the same frequency as scheduling decisions or, stated differently, at the same resolution. Moreover, it is unintrusive (generate as few extra system load as possible) because the generator is considered by the scheduler itself, at the same time as other processes.

This generation method has been made possible by two new features added in recent Linux kernel releases ($\geq$ 2.6.23): the `cpuset` and the `group scheduling` mechanisms. The `cpuset` feature enables programmers to attach to processes a set of CPU cores on which they can be executed. This restriction to a set of resources is enforced by the kernel itself. The `group scheduling` feature enables programmers to control resources sharing performed by the scheduler. When this feature is in effect, all processes necessarily belong to one control group. The scheduler distributes timeslices between groups in proportion to their priority. Within each group, classical priority scheduling is applied between processes to share assigned timeslices.

PPoPP'10, January 9–14, Bangalore, India.
ACM 978-1-60558-708-0/10/01.

[1] KRASH is available at `http://krash.ligforge.imag.fr` and provided free of charge under the terms of the GNU GPL license.

	General dynamic load profile	Side effects on scheduler	Arbitrary number of processes	Intrusiveness	Resolution
KRASH	Yes	Negligible	Yes	Negligible	Same as scheduler
Wrekavoc [Olivier Dubuisson and Jeannot 2009]	Not implemented	Low	No	Active poll	Higher than scheduler
Real time priority [Canon and Jeannot 2006]	Not implemented	High	Yes	Periodic wakeup	Poor
Frequency scaling [Srihari Makineni 2003]	No	Medium	Yes	Negligible	Higher than scheduler

Table 1: Features comparison table for several load generation solutions.

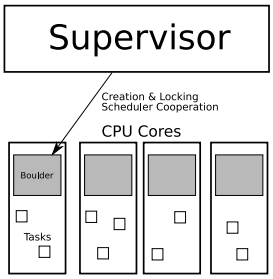

Figure 1: Diagram of our solution, based on boulder and supervisor processes

Taking into account theses features our generator is set in place by performing the following steps:

- create a new control group that we name the "base" group and move all processes into this group.
- for each CPU core, create a load generation process with minimal memory footprint (we name it the "boulder"), a new control group, attach the load generation process to this control group and restrict the control group to the concerned CPU core.
- choose a CPU core to run a supervisor process. This supervisor is given an input load profile and monitor load on all the load generation processes. Whenever required, it adjusts the priority of the groups that contain load generation processes.

Figure 1 gives a simple diagram of this method.

3. Evaluation

For our experiments we used a SMP system made of 8 Dual Core AMD Opteron 875 (2.2 GHz), 32 GB of RAM and a RAID 1 250 Go storage subsystem. This machine runs a Linux Debian Sid (unstable) Operating system with the latest kernel (version 2.6.30).

Figure 2 reports the load we measured when using KRASH (plain lines) to reproduce a custom reference profile (dashed lines) while a concurrent Linpack is running. This is only a part of the KRASH validation campaign we have conducted. It shows that our tool generates load with an average error of 2% and a standard deviation around 1%.

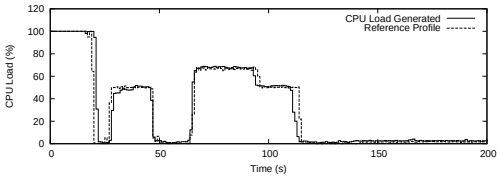

Figure 2: Load generated by KRASH in concurrence with Linpack

Regarding features, we summarize main characteristics of KRASH and other existing load generation method in table 1.

As none of the other solutions we know about is able to generate an arbitrary dynamic load profile we have compared KRASH with them regarding constant load generation only. Our main concern in this comparison is the obtrusiveness of each tool: a CPU-only load generation method might induce undesirable side effects on other resources in the system. System tasks that are the most affected by the generated CPU load are the ones that can usually be overlapped with computation tasks: networking operations and I/Os transactions. When hindering the scheduler efforts to nicely order running processes, a load generation method might prevent overlapping and induce worse than expected network or I/Os performance degradation. Conversely, methods that physically degrade target machine characteristics (the frequency scaling method for instance) might induce a better use of networking and computation overlapping opportunities. Both issues are outlined by the following experiment.

The benchmark we use in this evaluation is NAS NBP DT, which performs intensive point-to-point blocking MPI communications between all the involved processes. We have run this benchmark with 80 processes and a random communication topology. The slowdowns we have obtained on this benchmark when loading the machine at 50% with our different methods are reported in table 2. This experiment clearly indicates that previous suggested methods for CPU load generation can induce undesired side effects on other system resources such as communications. While not presented here, we have made several other experiments that validate this conclusion regarding I/O operations.

	Execution time		
	average	std dev.	Slowdown
None	2.9	0.5	1
KRASH	6.2	0.8	2.1
Wrekavoc	NA	NA	> 100
Real time priority	11.3	3.2	3.9
Frequency scaling	4.4	0.6	1.5

Table 2: Effects of load generation solutions on NAS NBP DT.

4. Conclusion

KRASH is able to generate in a reproducible way dynamic CPU loads and performs better than previously proposed solutions. It enables developers to validate the efficiency of their parallel applications in heterogeneous environments by generating an unintrusive and unobtrusive CPU load. While the CPU is usually the most important resource in high performance computing, some applications also heavily depend on other parts of the system such as caches, memory, I/O subsystem or network. Our future works will focus on extending KRASH to make it able to generate load on these other parts.

References

J.C. Canon and E. Jeannot. Wrekavoc: a tool for emulating heterogeneity. In *15th IEEE Heterogeneous Computing Workshop (HCW 06)*, 2006.

Jens Gustedt Olivier Dubuisson and Emmanuel Jeannot. Validating Wrekavoc: a tool for heterogeneity emulation. In *18th IEEE Heterogeneous Computing Workshop (HCW 09)*, 2009.

Ravi Iyer Srihari Makineni. Measurement-based analysis of tcp/ip processing requirements. In *In 10th International Conference on High Performance Computing (HiPC 2003)*, 2003.