Performance Monitoring in Distributed ParalleX Applications

Katelyn Kufahl[1], Bryce Lelbach[2], Hartmut Kaiser[2,3]

[1]Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, Wis.
[2]Center for Computation & Technology, Louisiana State University, Baton Rouge, La.
[3]Department of Computer Science, Louisiana State University, Baton Rouge, La.

Introduction

Moore's Law suggests that within the next decade, supercomputing systems will achieve performance in the range of exaFLOPS, nearly doubling the combined computational power of today's top 500 high performance systems[1]. Yet, despite progress in the capabilities of hardware, some classes of applications are limited in their scalability because of inefficiency in conventional parallel programming methods. Such scaling-impaired applications, like those used in scientific simulations that have non-uniform resolution, are unable to effectively use more than a few hundred processors[2].

The ParalleX model is designed to circumvent such problems by avoiding synchronized communication and implementing workload-balancing among processors at runtime. Eventually, the associated overhead will be reduced by new hardware solutions tailored to ParalleX.

Performance Counters

Performance counters allow for the benchmarking of HPX, the C++-based runtime system by which ParalleX is implemented. They function as an internal diagnostic tool in the development of HPX source code, because they expose the quantitative behavior of the system during or after execution of user applications.

Performance counters are written into HPX runtime source code, and any time-dependent data collected during execution of a user application is done using an HPX application titled “Heartbeat”.

References

[1]*Number of Processors share for 06/2011*
TOP500 Supercomputing Site
http://www.top500.org/stats/list/37/procclass

[2]H. Kaiser, et al., *ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications*
ICPPW '09 Proceedings of the 2009 International Conference on Parallel Processing Workshops, 394-401

Results

Figure 1 exhibits the behavior of the thread management subsystem as measured by performance-counting code in the main thread manager loop. When a PX-Thread is not being executed, the thread manager performs maintenance such as creating and scheduling new threads into a queue.

The data in Figure 1 was collected while the system was work-starved, because the user application required too little computation for the number of localities in the cluster. The data thus suggests that increasing the number of processors only increases overhead of the system when the computational problem is too small. The source of the oscillatory behavior and “flooring” effect is currently unknown and in need of further investigation.

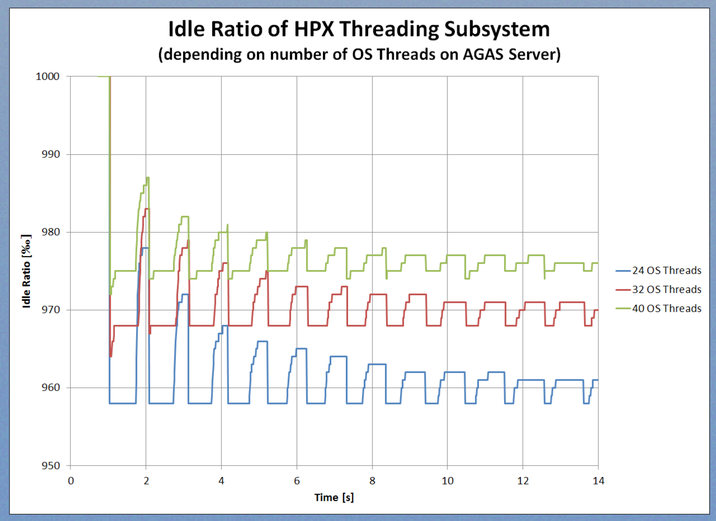

Figure 1: Behavior of Thread Management Subsystem in a Work-Starved Environment

Further Information

On the World Wide Web:

http://px.cct.lsu.edu

Contact the ParalleX group:

gopx@cct.lsu.edu

Implementation

The principles of design around which HPX is built are:

- *The Active Global Address Space (AGAS)*

  Dynamic workload-balancing among localities in a cluster necessitates a common address space across the system.
- *Local Control Objects*

  To organize flow, specialized synchronization primitives such as LCOs are required. For example, a *future* is an LCO that postpones a calculation until the value is needed.
- *Parcel Transport and Parcel Management*

  Messages are exchanged between localities by parcels, and carry the work to the data.
- *PX-Threads and Thread Management*

  User-level threads called PX-Threads are scheduled into work queues managed by operating system (OS) threads..

Future Work

There are dozens of possibilities for other useful performance counters and their statistical analysis in HPX, including but not limited to:

- Work-queue lengths in the Active Global Address Base
- Average time elapsed between the sending and receiving of a parcel between localities
- Waiting time for PX thread-queues in the thread manager
- Variance in thread-queue lengths when aggressive work-stealing is enabled versus disabled

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant OCI-1005165.