Performance Studies Towards Adaptive Thread Scheduling

Patricia Grubel[1], Jeanine Cook[1,2], Hartmut Kaiser[3,4]

[1]NMSU, Electrical and Computer Engineering, [2]Sandia National Laboratories

[3]Center for Computation and Technology, [4]LSU Department of Computer Science

Introduction

High Performance ParalleX (HPX) is an experimental parallel and distributed runtime system that implements the ParalleX execution model [1] on conventional SMP clusters. The traditional Communicating Sequential Processes (CSP) is replaced with asynchronous fine- and medium-grained parallelism in HPX.

HPX is implemented as a modular framework with four primary modules: the HPX Thread Manager, the Active Global Address Space (AGAS), Local Control Objects (LCOs) and the Parcel Transport Layer. Exploring dynamic optimization of these modules on conventional architectures will provide designs for future hardware architectures. The Performance Monitor supplies performance counters for the modules and hardware counters through an interface to PAPI.

Our objective is to determine if adaptive management of scheduling policies can be used dynamically to improve task management among hardware threads on a node. Once metrics have been determined, we will pursue applying them to dynamically change parameters of the thread scheduling policy for improved performance and efficiency of scaling-impaired parallel applications.

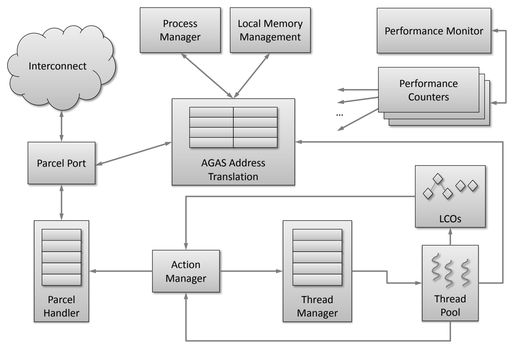
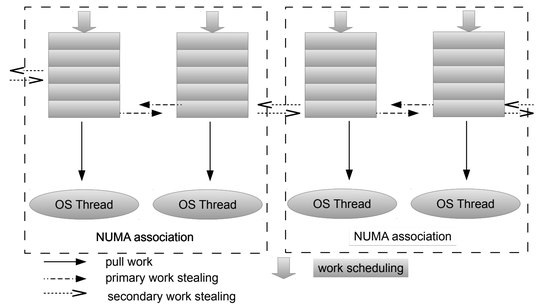


Figure 1: HPX Architecture

Thread Schedulers

Priority Local Scheduler (default)

• One FIFO Queue per OS Thread



Priority ABP Scheduler (priority_abp)

• One Double Ended Lock Free Queue per OS Thread

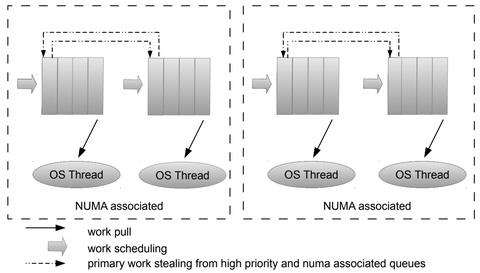
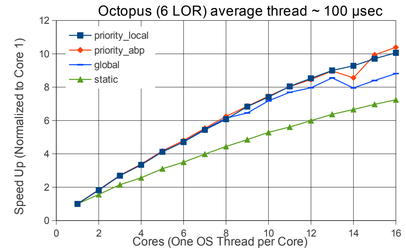


Figure 2: Work Stealing Thread Schedulers

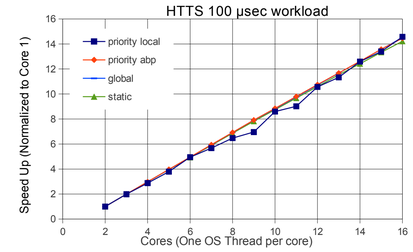
Performance Study

We conduct performance studies of various HPX thread scheduling policies using a task scheduling micro-benchmark, HTTS, the Unbalanced Tree Search (UTS) benchmark [2], and Octopus [3], an adaptive mesh refinement (AMR) application, on a variety of platforms to determine metrics which will aid in improving scheduling policies. Thread scheduling policies shown in Figure 2 use work stealing. Studies also use the global scheduler (a single work queue) and the static scheduler (one queue per OS thread, no stealing). Platforms used are Xeon E5 2690 (plots 1-4 & 6), Sandybridge, and Xeon Phi (plot 5).

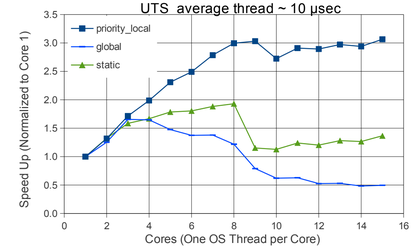
(1)



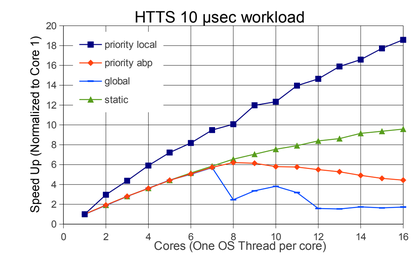
(2)



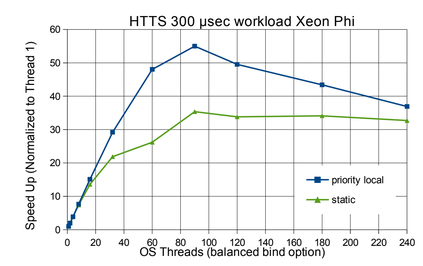
(3)



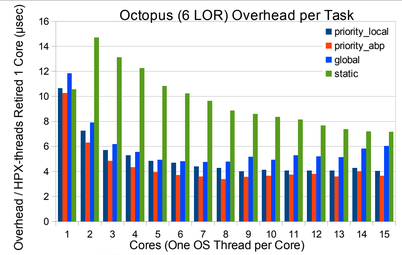
(4)



(5)



(6)



Results

Plots (1-4) illustrate strong scaling on one node for the three benchmarks. All data from Octopus was taken from the solver portion of the AMR application. The average thread length for UTS and Octopus runs are calculated using performance counters. We run HTTS with a workload comparable to those calculated. The overhead plot above shows an example of measurements derived from HPX performance counters.

The Priority ABP scheduler does not work for the UTS benchmark, which has fine-grained workloads, and performs poorly with similar workloads for the micro-benchmark.

Performance counters for these data sets were evaluated. HPX counters did show correlation to the decrease in scaling. Hardware counters had little correlation to the behaviors. Counters collected were aggregates for all OS threads for each run. Further studies should be made periodically measuring counters for each OS thread.

Acknowledgement

Vinay Amatya, Bryce Lelbach,Thomas Heller
XSEDE allocation 130032
NSF Grant CCF-111798 & Sandia Graduate Fellowship

Bibliography

[1] H. Kaiser, M. Brodowicz, and T. Sterling, “ParalleX: An Advanced Parallel Execution Model for Scaling-Impaired Applications,” in Proceedings of the 38th International Conference on Parallel Processing, ICPP '09, pp. 394-401, 2009.

{2] Stephen Olivier, Jun Huan, Jinze Liu, Jan Prins, James Dinan, P. Sadayappan, and Chau-Wen Tseng. “UTS: An Unbalance Tree Search benchmark.” In *Proceedings of the 19th international conference on Languages and compilers for parallel computing* (LCPC'06), 2006.

[3] github.com/STEllAR-GROUP/octopus