Weighted Distribution for Random Victim Selection in Distributed Work Stealing
Swann Perarnau, Mitsuhisa Sato
RIKEN AICS, University of Tsukuba

Distributed Work Stealing
- Algorithm for dynamic load balancing.
- Increasingly popular in shared and distributed memory.
- Work is divided in items or tasks.
- An idle process steals work from a busy one.

Workstealing Principle
while not finished do
  while task ← getWork(myStack) do
    task.run()
  end while
while myStack is empty do
  v ← selectVictim()
  steal(v)
end while

K Computer
- Each node is composed of one SPARC-VIIIfx with 8 cores.
- 80000+ compute nodes, in 800+ racks.
- Tofu proprietary network: 6D mesh torus.

Advantages of Work Stealing
- Fully distributed.
- Most of the scheduling overhead occurs on idle processes.
- Provably efficient.

Issues of Work Stealing on Distributed Systems
- Traditionally assumes uniform access times between processes.
- Ignore data transfers costs.
- Load balancing performance degrades at very large scale.

Our Work
- Change the victim selection process to improve average search time.
- Use network topology knowledge.
- Large scale execution on the K Computer.

Tofu Network Configuration
Figure: Coordinate system in a Tofu network: A,B,C inside a Tofu unit, X,Y,Z between them

A New Victim Selection Process
Most implementations use a random selection process.
- Uniform probability to steal each process.
- Provably efficient for shared memory systems.

Our Idea: use network topology information to weight the probability of a steal.
- Still a random, efficient selection process.
- Ensures eventual discovery/balancing of work.
- Compensate for response latencies between nodes at large scale.

$$ p(i,j) = \frac{w(i,j)}{\sum_j w(i,j)} w(i,j) = \begin{cases} 1 \cdot \frac{\text{Euclidean distance between } i \text{ and } j}{10^3} & \text{if } e(i,j) \neq 0 \\ 1 & \text{if } e(i,j) = 0 \end{cases} $$

UTS Benchmark Performance

Experimental Setup
- Public, pure MPI 2 implementation of UTS.
- UTS input is a tree with 157 billions of nodes.
- Default parameters for benchmark and platform scheduler.

Additional Measurements: 1 MPI Process Per Node

Acknowledgements
Results were obtained by access to the K computer at the RIKEN AICS.
This work was supported by the JSPS Grant-in-Aid for JSPS Fellows Number 24.02711.

RIKEN Advanced Institute for Computational Science
Programming Environment Research Team
perarnau@riken.jp