Scalable Work Stealing
Cluster and Grid Computing Seminar

Benedikt Hupfauf, University of Innsbruck
Content

Introduction

Scalable Work Stealing

Evaluation

Summary
Introduction

To solve a computational intensive problem

- Split problem into several independent tasks (if possible)
- Divide workload on multiple machines
- Collect results
Task Farming (Master/Worker Approach)

- One master, multiple workers
- Master coordinates execution of tasks
- Not scalable!
Distributed Task Pools

- Every process maintains its own work queue
- Load balancing: idle processes try to steal work
- Work Stealing is receiver-initiated
- But: Synchronization needed!
### Split Queue

- Normal queue needs to be locked before every access
- Locking is expensive!
- Split queue into two sections, private and public
- Private section: only owner has access
- Public section: shared, every process has access
- Only public section needs to be locked
Randomized Work Stealing - Algorithm

1. Schedule task from work queue, if empty continue with 3
2. Execute task, when done continue with 1
3. Check for termination, if terminated continue with 5, else 4
4. Select random victim and try to steal work, if not possible go to step 3
5. Done.
Figure: Initial state
Figure: Worker 1 finished task
**Figure:** New task is scheduled
Figure: Worker 1 finished task, new task is scheduled
**Figure:** Worker 1 finished task, no more tasks available
Figure: Not yet terminated, steal tasks from random victim
Scalable Work Discovery

Consider the following example

- Assume simple synchronization algorithm (e.g. Bakery algorithm)
- Two processes want to steal work
- First one acquires lock, second one has to wait
- When the second one acquires the lock there is no work to steal

Even if a process detects that there is no work left to steal it has to wait until it acquires the lock! Solution: more flexible synchronization algorithm that allows abortion.
Synchronization

Locking Strategies

- **ARMCI locks** simplified bakery algorithm
- **Spinlocks** busy waiting
- **Aborting Steals** frequently check, while waiting, if work is available
Benchmarks

Problems with respective challenges

- **Madness**  size and shape of task tree are unpredictable
- **Bouncing Producer-Consumer**  locating the work is challenging
- **Unbalanced Tree Search**  amount of work per task varies significantly
Figure: Efficiency for MADNESS 3D tree creation kernel
Bouncing Producer-Consumer Benchmark

Figure: Efficiency for Bouncing Producer-Consumer benchmark
Figure: Efficiency for Unbalanced Tree Search benchmark
Summary

What has been discussed?

• Task Farming, dividing a problem into smaller ones
• Work Stealing, for balancing workload
• Split Queues, to reduce lock overhead
• Aborting Steals, to reduce unnecessary waiting
• Evaluation using three worst case examples

Randomized work stealing in combination with split queues results in a highly efficient and scalable system, with low overhead.
Scalable Work Stealing
**Split vs. Lock Queue**

**Figure:** Performance of different queuing strategies
How much work should be stolen?

![Graph showing performance of different stealing strategies](image)

**Figure:** Performance of different stealing strategies