Scheduling Multithreaded Computations by Work stealing

by Robert D Blumofe and Charles E. Leiserson, MIT Laboratory for Computer Science

Presented by Dayton Bishop on
May 3rd, in the Software Systems Seminar 2007
Department of Computer Sciences, University of Salzburg

Overview

- Introduction to the thematics of Multi Processing
- Goal of the presentation
- Work Sharing
- Premises for work stealing
- Steps towards the development of the work stealing Algorithm
- Analysing the algorithm
- Comparison of the algorithms Work Sharing and Work Stealing
- Improvements
- Implementation

Introduction to the thematics of Multi Processing

- Multiprocessor systems have been around for some time
- Single processors can only increase their speed according to Moore’s law
- At some point there is a limit of the processor speeds
- Single processors are more expensive than multiprocessors for same processing power
- Uniprocessors are easier to handle from a software perspective

Software on Multiprocessing systems

- Design issues:
  - Additional functionality
  - High requirements to the operating system to hide the multiprocessing system
  - Communication Overhead
  - Multiprocessor scheduling is a NP complete problem
  - Therefore a good simulation of the problem is the bins packing problem

Constraints of Multi Processing

- Factors limiting processing power
  - Temperature
  - Communication Speeds
  - Power consumption
  - Software

Goal

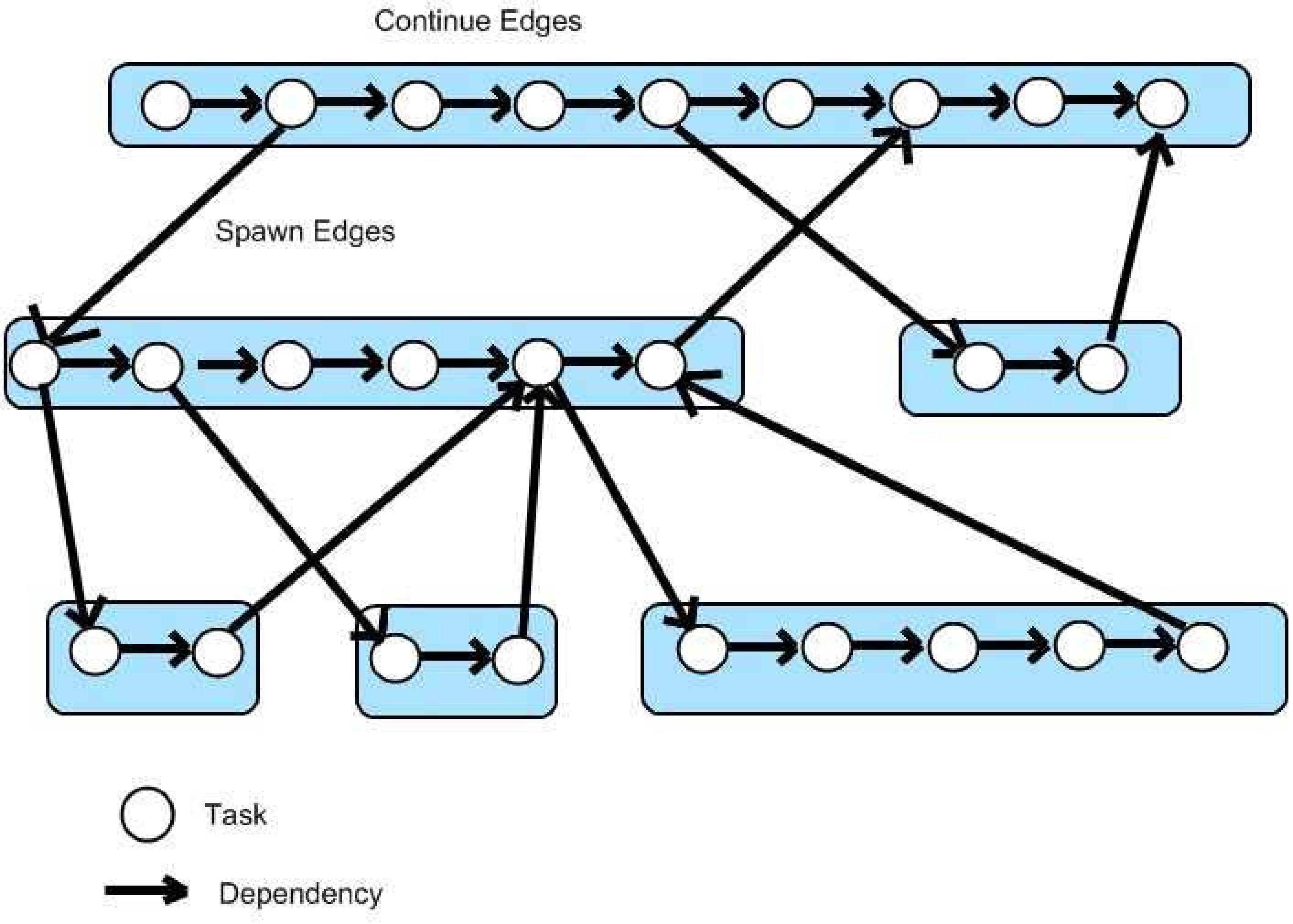
- Comparing the known alternatives of
  - Work stealing
  - Work sharing
- Proving that Work Stealing requires less communication than Work Sharing
  - Work Sharing Communications : $\Theta (T_1 S_{Max})$
  - T_1 : Minimum execution time for one processor
  - S_{Max} : Size in Bytes of the largest activation frame

Work Sharing

- General Idea is a Global queue
- There are other papers that propose a distributed shared work queue
- Each processor requests a Thread to work on from this central queue
- If the thread is stalled it is returned to the central queue

Premises on Threads

- Life of a thread:
  - Spawn
  - Stalls
  - Dies
- DAG
- Fully Strict
- Example for a Uniprocessor execution


Continue Edges
Spawn Edges
Task
Dependency

Premises on Threads (2)

- Heavyweight threads
  - Activation Frame
  - The frame hold all values
  - No global storage
- A parent with Children remains alive
- Activation Depth S_1= Minimal Amount of space possible
  - Total size of all frames of the execution
- S_P Linear expansion of space for a P – Processor execution schedule

Terminology

- T_P : Time used by a P – processor execution schedule
- $T_{infinity}$: Time of computation for an infinite amount of processors
- $T_P >= T_{infinite}$
- Work : The number of tasks in the computation
- T_1 : The minimum time for a Uniprocessor
- $T_P >= T_1 / P$

Steps twards the comparison

- The Greedy Scheduling Algorithm
- The Busy Leaves Algorithm
- Randomized work – stealing algorithm
- Refining using the atomic access model and recycling

The Greedy Scheduling Algorithm

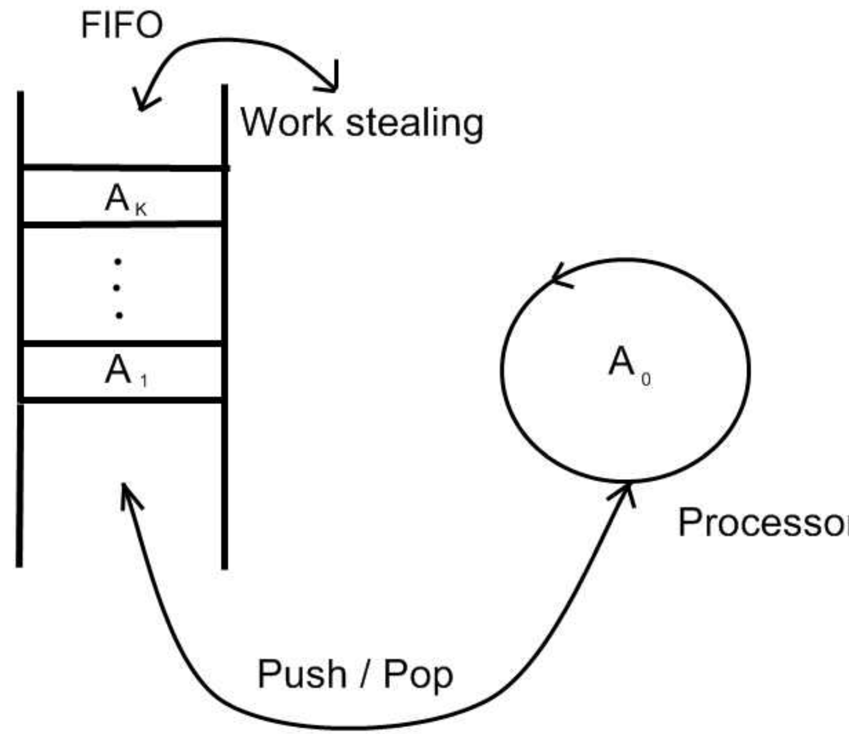
- Linear Speed up
- If P tasks are ready P execute
- If less are ready all execute

The Busy Leaves Algorithm

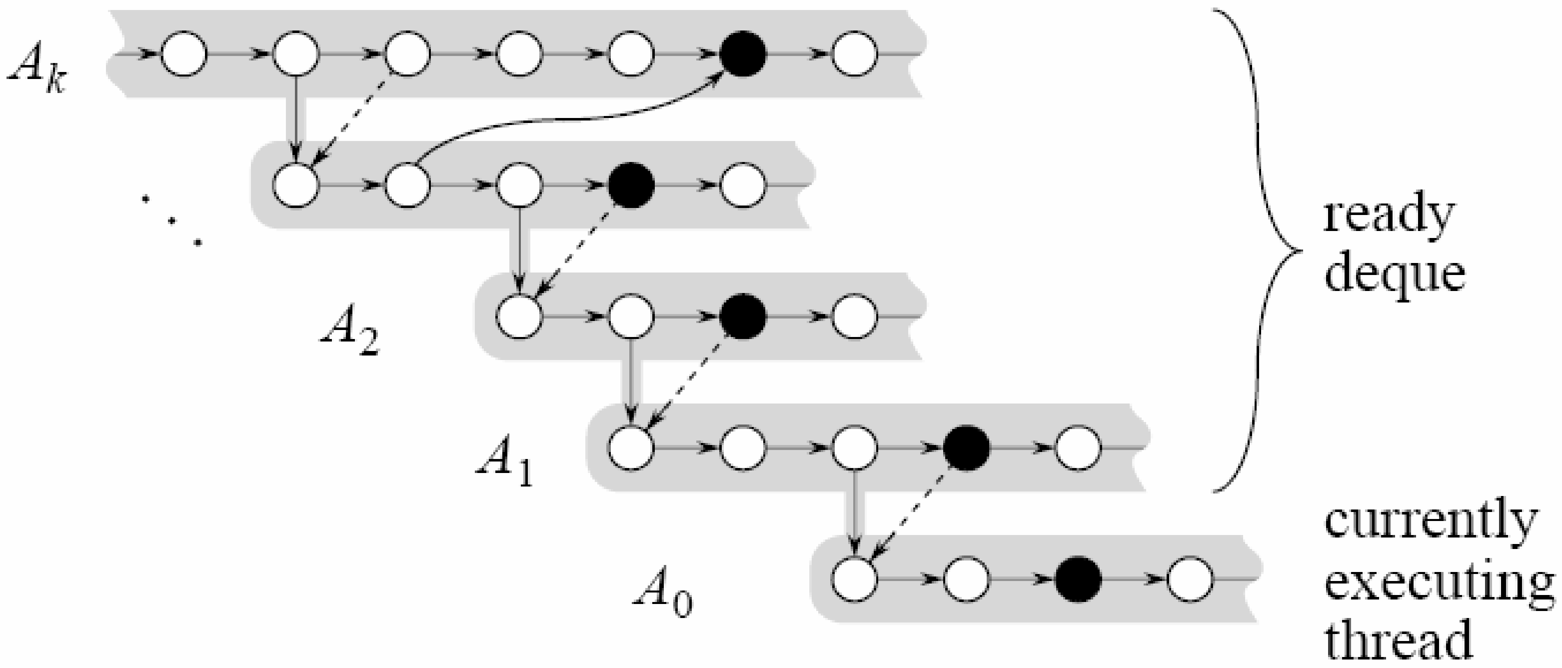
- No contending for access to thread pool
- Unit access to the pool
- Average available Prallelism: $T_1 / T_{infinity}$
- Good for small scale systems
- No scaling to large scale systems
- Operation:
  1. A spawns B, Then B is executed
  2. If A stalls, return A to the thread pool
  3. If A dies, Parent is executed. Else other thread in pool
- $S_P <= S_1 * P$

Randomized work – stealing algorithm

- Each processor maintains a Thread deque
- Maintains the busy leaves properties
- Actions
  1. If Thread A enables Thread B, A is placed in the ready queue
  2. If A spawns B, B is executed
  3. If A dies, no Threads -> work stealing from random processor



Randomized work – stealing algorithm



The atomic access model and recycling

- Balls and Bins Game
- P - Balls
- P - Bins
- M is the number of the Requests
- Balls are tossed randomly into bins
- Rules:
    1. Random balls from reservoir into Bins
    2. Removes one ball from each Bin

The atomic access model and recycling

- Game ends after
  - M ball tosses
  - All Balls have been returned to the reservoir
- Ball symbolizes a steal request
- Interest: Total delay time
  - Remains rather small

Balls and Bins Game Results

- Expected total delay = The number of Requests
- Start with analysing 1 ball
- Either it is delayed or it is not delayed at every of the m throws
- Does not matter what ball is removed from the bin first
- Then we assume that all the balls are equal
- The total delay is the sum of P delays of balls

Analysing time and communication cost

- An accounting argument was used
- Each round P dollars are available
- These dollars are then distributed among three bins:
  - Wait
  - Steal
  - Work
- The execution finishes when as many tokens are in the work bin as there are tasks
  - At the end there are task tokens in the work bin

Analysing time and communication cost

- Number of calculated dollars in the steal bucket:
  - P times the longest Path of the DAG
- Number of calculated waits:
  - Is at most the number of dollars in the steal bucket
- The total communication for work stealing:
  - The number of steal attempts times the amount of information S_{Max} -> $O(P * T_{infinity} * S_{Max})$
  - Since in linear Speedup systems we assume $P = O(T_1 / T_{infinity})$
- Result for the communication : $O(T_1 S_{Max})$

Comparison of the algorithm to other methods

- Work sharing
  - Work Sharing Communications : $\Theta (T_1 S_{Max})$
- Work stealing
  - $O(T_1 S_{Max})$
  - Since $P << T_1 / T_{infinity}$ we expect much better results

Improvements

- No guarantee that processors can run out of space
- Working with strict not fully strict graphs
- Possibly even non strictness

Implementations

- Cilk
  - C implementation for Multiprocessors of Work stealing
- Achievments:
  - Chessprogramms that have won awards

Danke für die Aufmeksamkeit

Scheduling Multithreaded computations by work stealing

Presented by Dayton Bishop

Ressources

1. Robert D. Blumofe and Charles E. Leiserson. Space-efficient scheduling of multithreaded computations. In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, pages 362–371, San Diego, California, May 1993.

Proof(1)

1. Delay D
2. Delay of one ball by another
3. Probability of for a delay
4. Proving Inequality

$$D = \sum_{r=1}^{P} \delta_r \ . \tag{1}$$

$$\delta = \sum_{i=1}^{m} \sum_{r=2}^{P} x_{ir} \ . \tag{2}$$

$$\Pr\left\{ \bigwedge_{(i,r)\in S} (x_{ir} = 1) \right\} \le P^{-|S|} \ . \tag{3}$$

$$\Pr\left\{ x_{ir} = 1 \,\middle|\, \bigwedge_{(i',r')\in S'} (x_{ir} = 1) \right\} \le 1/P \ , \tag{4}$$

$$\begin{aligned} D &= \sum_{r=1}^{P} \max\{2em_r, \lg P + \lg(1/\epsilon)\} \\ &= \Theta(M + P\lg P + P\lg(1/\epsilon)) \ , \end{aligned}$$