Multiprocessor OS

ECE 447 – Intro to Computer Architecture
Spring ‘10

Prof. James C. Hoe
Prof. José F. Martínez

CC-NUMA

- Large-scale design point brings very diverse workload mix
  - size
  - nature
- Memory hierarchy complex, large impact on performance
- OS faces complex choices
  - process scheduling
  - page migration/replication
Process Scheduling

- Given \( n \) applications, each app. \( i \) with \( m_i \)
  - what threads to schedule?
  - where?
- Different approaches
  - affinity scheduling
  - gang scheduling
  - processor sets
  - process control

Affinity Scheduling

- Data-driven mechanism
- Cache affinity: schedule process where it ran last
  - exploit cache locality
- Cluster affinity: schedule process in the vicinity of where it ran last
  - maintain locality of cache misses
    - neighbor communication
    - cache-to-cache transfers
  - exploit remote access cache
- Can be implemented in Unix by tweaking priorities at schedule time
- No notion of belonging to a multi-threaded application
  - can be combined with other policies
Gang Scheduling

- Communication-driven mechanism
- All threads of an application scheduled to run at the same time
  - or as many as possible if \( \#\text{threads} > \#\text{processors} \)
- Inspired by \textit{convoying} effect in busy-wait synchronization
- Can be implemented using a “row” schedule layout
  - multiple rows if \( \#\text{threads} > \#\text{processors} \)
- Scheduler decides on many processors at the same time
  - restricts flexibility
- Favors large applications
  - Can compensate for that by prioritizing

Processor Sets

- Resource-driven mechanism
- Partition machine, assign each application to one partition
  - possible \( \#\text{threads} > \#\text{processors} \) in each partition
- Reduces cross-application interference
  - cache
  - local memory/remote access cache
- Does not (necessarily) favor large applications
  - Large apps typically handled during “special” time slots
Process Control

- Efficiency-driven mechanism
- OS instructs application to adjust #threads to reach best operating point
- Goal: maximize efficiency system-wide
  - minimize processor waste
- Amenable to task-queue application model
  - transparent to programmer

Chandra et al., ASPLOS94

- Affinity scheduling reduces miss rate
- However, large fraction of remote misses remain
  - need to bring adapt page allocation to application needs
Page Migration

- Ratio between remote and local mem accesses
  - CC-NUMA: 3-5-10 times
  - CC-NOW: 10-20 times
- Data locality potentially most important issue in CC-NUMA
  - Compounded in time-shared servers
- What to do?
  - Page migration: Follow the process
  - Page replication: For read-shared pages

Problem Statement

- Minimize runtime by reducing component of mem stall
  - Convert remote accesses to local
  - Not suffer from excessive OS overhead
- Approach: find patterns
  - Single process: possibly migrate
    - Sequential app code or data
    - Disjoint data of parallel app
  - Read-mostly: possibly replicate
    - Code (parallel or concurrent seq apps)
    - Read-mostly data
  - Read-Write: generally not candidates
Cost of Migration/Replication

- **Bookkeeping**
  - Count or sample cache misses, TLB misses, etc.
- **Kernel overheads**
  - Allocate new page
  - Change physical mappings
  - Flush TLBs
  - Eliminate replicas (if needed)
- **Data movement**
- **Increased memory use**

---

**Decision Tree (on Cache Misses)**

Miss Rate To Page?

- **Low: Do nothing**
  - High: Write freq. Mem pressure?
    - High: Do nothing
    - Both Low: Replicate
  - Low: Migration rate?
    - High: Do nothing
    - Low: Migrate
- **High: Sharing?**

Hoe & Martinez
Results of Verghese et al.

- OS-induced page migration+replication: 29% performance gain
  - Over first-touch policy
- Both migration, replication needed
- TLB misses not sufficient to decide policy
- Kernel needs modifications to keep overhead low
  - Risk of negating performance improvements