Scheduling

Jesus Labarta

Applications submitted to system \( \rightarrow \) Resources x Time

- Resources:
  - Processors
  - Memory
  - ...

Objective

- Maximize resource utilization
- Maximize throughput
- Minimize response time

Not necessarily the same
Scheduling

- Applicable policies depend on
  - Application structure
    - Fixed
    - Moldable
    - Malleable
  - Resources
    - Level of sharing support
    - Level of configurability

Malleability

- Dynamically adaptable parallelization structure
- Responsiveness to
  - Resource availability: Processors, memory, ...
  - Application behavior dependence on input data, time, ...
- Malleability requires:
  - Separation between
    - Algorithm: Problem logic. Full responsibility / Job of the programmer
    - Scheduling: Efficiency Responsibility of the system. Programmer hints
  - Frequent control points
- Issue of:
  - Programming model support → OpenMP
  - Programming practices
### Programming practices and malleability

#### Scheduling decisions: Once for all

Explicit code only related to parallelism

```
C$OMP PARALLEL
WhoAmI=RunTimeCall()
myBlock=f(WhoAmI)
...
Call Compute1(myBlock)
...
DO iters=1, #iters
    Call Compute2(myBlock)
END DO
...
C$OMP END PARALLEL
```

```
C$OMP PARALLEL DO
    DO Block=1, #blocks
        Call Compute1(Block)
    ENDO
C$OMP END PARALLEL
...
DO iter=1, #iters
    C$OMP PARALLEL DO
        DO Block=1, #blocks
            Call Compute1(Block)
        ENDO
    C$OMP END PARALLEL
END DO
...
C$OMP END PARALLEL
```

```
myBlock ∈ [ 1,#processors] ← f(resources)
Block ∈ [ 1,#blocks] ← f(algorithm)
```

---

### Scheduling

#### Hierarchy
- Grid
- Jobs scheduling
- Process scheduling
  - Computation task
  - Processes
  - User threads
  - Kernel threads
  - Processors

#### Typical
- Very simple approaches at many levels
  - 1 to 1
- One most relevant level
- Lack of coordination

Multiprogrammed workload

Algorithm

User level threads

Kernel threads

Physical resources

Jesús Labarta, MP, 2002
Job scheduling

- Control of Multiprogramming level
  - Static
  - Dynamic

- Long term scheduling (minutes – hours – days)

- Coarse grain allocation of resources

- Basic operation modes
  - Batch queuing systems
  - Interactive (≡ no job scheduling)

Job scheduling

- Basic approaches
  - Fixed Partitions
    - Different sizes of processors, memory
    - One queue per partition
    - Internal fragmentation
  - Variable partitions
    - Single queue
    - External fragmentation
Job scheduling

- Basic policies
  - FIFO
  - SJF
  - Backfilling

- Requirements
  - Estimates of job duration
  - Estimates of job performance vs. resources
  - User supplied / prediction

Job scheduling

- Typical setups
  - Different queues
    - Memory, duration,…
  - User submits to queue that fits estimate of job needs
  - A lot of manual supervision by operator, modifying job limits,…. 
Processor scheduling

- Mapping of kernel threads to processors

- Basic approaches
  - Time sharing
  - Space sharing
  - Gang scheduling

- Types of policies
  - Fixed
  - Adaptive
  - Dynamic

Time sharing
- Extensions from sequential
- Multiprocessor specific effects/features
  - Applied to individual kernel threads
    - Fairness issues ➔ fair share schedulers
  - Load balance vs. locality trade off
    - Local queues
      - Binding / Migration
    - Global queues
      - Contention / Affinity
Processor scheduling

- Time sharing
  - Multiprocessor specific effects/features
    - Throughput vs. response time
      - Capability to fill all gaps \( \rightarrow \) more threads than processors
      - Interaction between OS scheduling and application synchronization \( \rightarrow \) high risk of inefficiency \( \rightarrow \) threads == processors

Motivation example

- Sweep3d 4 MPI tasks with 4 OpenMP threads each
  - 16 CPUs SMP
  - XLMSOPTS=spin=0;yield=0

![Diagram showing main threads and Slave OpenMP threads]
Motivation example

- Sweep3d 4 MPI tasks with 4 OpenMP threads each
  - 16 CPUs SMP
  - XLMSOPTS=spin=0:yield=0

- + 1 sequential program
  - Non instrumented

Quantum == 10 ms
Intricate precedence and scheduling interferences

2 threads time sharing CPU

Can not finish because receiver descheduled
Receive called after finishing send
Can not finish because sender descheduled

Same scale as previous slide

Jesús Labarta, MP, 2002
Motivation example

- Thread scheduling
  - XLMSOPTS=spin=1;yield=1
  - CPU bursts << Quantum
  - Lots of process migrations of OpenMP slave threads

Processor scheduling

- Space sharing
  - Partition of processors among applications
    - Possible fragmentation
  - Example policies
    - Equipartition
    - Equal efficiency

- Gang Scheduling
  - Coarse grain time sharing
**Equipartition**

- Equal allocation among running jobs [McCann93]
  - Allocation = P/num_applications
- Re-allocation at each job arrival and completion
- Reasonable approach if job performance not known

**Equal_efficiency**

- Dynamic space-sharing policy [Nguyen96]
- Job performance analysis + Eff. extrapolation
- Assigns processors (one per turn) to those applications that achieve the higher efficiency
Gang Scheduling

- Threads are grouped into gangs
- Threads in a gang are executed simultaneously
- Time-sharing is used among gangs

![Gang Scheduling Diagram]

NANOS OS scheduling environment

- OpenMP Parallel Applications (malleable)
- Queuing System
- CPU Manager
- Operating System
- Shared Memory
- Multiprocessor

![NANOS OS Scheduling Diagram]
Native time sharing vs. PDPA

- 3 BTs x 8 processes @ 16 CPUs

Page placement

- Mapping of pages to nodes
- UMA
  - Not an issue
- NUMA
  - Round Robin - Random
  - First touch
  - Page migration
Page placement

- Should the programmer specify data distribution?
  - Difficult problem → Automatic

- Approach
  - Application run time measures
  - Computes “good” distribution
  - Asks the system to redistribute

- Provides
  - Responsiveness
  - Local view → Stability

Page placement & OS scheduling

Native page migration enabled

- Data can’t follow computation

disables

- Data follows computation

Jesús Labarta, MP, 2002