UNIVERSIDAD DE BURGOS

Parallel Architectures & Programming

José M. Cámara
(checam@ubu.es)

v. 1.0

UNIVERSIDAD DE BURGOS

Agenda

Three different approaches.

Operating system level.

- Network operating systems.
- Multiprocessor operating systems.
- Distributed operating systems.
  - Message passing systems.

Application level programming.

- Concurrent programming.
  - Synchronization in shared memory systems.
  - Synchronization in message passing systems.
  - Management of concurrent processes.
  - Concurrent programming motivation.

Management level.

- Scheduling.
- PBS.

Relevant issues.

Program: piece of code to be executed by the processor (machine code).

User: the one to make use of the computer (programmer, administrator,...).

Operating system level

Operating systems for shared memory multiprocessors:

- **Tightly** coupled software
- **Tightly** coupled hardware

Sistema operativo de red (actualmente todos):

- **Loosely** coupled software
- **Tightly** coupled hardware

Distributed operating system (DOS):

- **Tightly** coupled software
- **Loosely** coupled hardware

In the past, there were a number of OS available for MIMD machines (usually vendor Unix distributions). More recently, most of them have been dismissed and merged into a few Unix vendor distribution and, more commonly been replaced by Linux. In the last few years MS Windows has found a place in this market as well.

UNIVERSIDAD
DE BURGOS

Network operating systems.

Make operations on remote computers possible by the use of the network adaptor.

Currently, all operating systems provide this functionality, so we will not deepen in their study.

They make parallel computation feasible but not very efficiently.

UNIVERSIDAD
DE BURGOS

Multiprocessor Operating Systems I

Like network OS, their capabilities are available in all modern systems.

They make multiprogramming management over multiprocessor hardware possible.

What is different from single processor management is that there is a hardware resource, the processor, that has been replicated.

It is the scheduler's job to manage the queue of awaiting processes and make an efficient use of all available processors.

These OS are designed to manage both multiprocessor and multicore systems.

UNIVERSIDAD
DE BURGOS

Multiprocessor Operating Systems II

Two important concepts:

- Process: protection and resource allocation unit.
- Thread: scheduling unit.

A single process may host several threads. Each thread has its own stack, state of execution and processor context.

Threads belong to both user application programs and operating system. In this regard there are:

- Symmetric multiprocessors: all nodes have the same functionality and are equally able to execute OS routines.
- Asymmetric multiprocessors (master – slave): one node executes OS and the rest only deal with programs -> A high number of slaves may collapse the master.

In this scenario, race conditions when the OS is concurrently executed may arise. Access to the threads queue from concurrent scheduler routines have to be performed in mutual exclusion conditions.

UNIVERSIDAD
DE BURGOS

Multiprocessor Operating Systems III - Scheduling

Processor (thread) scheduling in multiprocessors involves:	Allocating CPU time to processes.
	Decide which CPU will host the process.

There are various models:	Time sharing: all threads are queued and are independently scheduled.
	Space sharing: related threads (belonging to the same process or task) are scheduled together; they are allocated when a sufficient number of CPUs are available and are never interrupted.
	Gang scheduling: related threads can be interrupted but they are all stopped and resumed simultaneously.

UNIVERSIDAD DE BURGOS

Multiprocessor Operating Systems IV - Example

LINUX includes multiprocessor support from 2.6 kernel. O(1) scheduler is then implemented.

Process scheduling is time shared. Two process queues are created on each CPU: one for active processes and one for the expired ones.

Processes join the “expired queue” when they use up their time slot.

For a certain priority level, when all its tasks are expired the queue turns active.

Each queue handles 140 priority levels, being the upper 100 for real time tasks. Within each priority level the queue is a FIFO one.

Every 200ms processors’ load is monitored and balanced.

The one queue per CPU scheme improves cache efficiency. Retrieving a previously issued task decreases cache misses.

UNIVERSIDAD DE BURGOS

Distributed operating systems

They make a system integrated by independent computers work, in the eyes of the user, as a single (virtual) machine.

- Modern OS do not include this functionality.

But there are various ways to implement it:

- Client server architectures: implemented, for instance on information systems, where local clients access remote servers by the use of a query language.
- Remote procedure calls: application programs make calls to procedures that are executed in remote systems.
- Message passing systems: processes executed in different computers exchange messages and collaborate to carry out a common task. This is the most common option in parallel computing systems.

It is necessary to install an additional software layer called middleware. It will provide virtual machine capabilities, involving two main aspects:

- Process migration: it must be able to move processes from one processor to another in order to, for example, achieve load balancing.
- Fault tolerance: must respond faultlessly to the disappearance or incorporation of nodes at runtime.

UNIVERSIDAD DE BURGOS

Message passing systems

Implementation alternatives for message passing:	
	Reliable: ensures message delivery and provides mechanisms to send acknowledgement to the sender.
	Unreliable: injects the message into the network and forgets it. It is a less secure mechanism but also less costly in terms of network overloading.
	Blocking: interrupts requesting process until the requested operation has been completed (send or receive). Completion does not imply delivery; just a copy of the message in a local buffer is enough.
	Non blocking: processing continues regardless of the situation of the message. Communication is completed in background. It is faster but may lead to race conditions.
	Synchronous: interrupts sender process until the receiver has collected the message.
	Asynchronous: sender interruption depends on whether the system is blocking or not.

The middleware may provide just one of the former options or let the user choose among several or even all of them.

There are two possible strategies when implementing the middleware:	
	Extend the operating system providing distributed system capabilities.
	Integrate in the software stack as an intermediate layer . It is not part of the operating system but provides an API for user application programs (MPI).

Both strategies may be complementary: the first one is usually a better approach to the definition of virtual machine, whereas the second make the development of parallel programs possible and provides a basic approach to the capabilities of a virtual machine.

Programming level

There are two paradigms associated to parallel programming:	Implicit parallelism: the user is not committed with hardware exploitation. On the contrary, he relies on lower software layers (compiler, OS) to do that job for him.
	Explicit parallelism: the user gets involved in the development of parallel application programs under any of the options available for him.
Explicit parallelism models:	Traditional languages with parallel libraries (MPI). It requires a certain start up effort and high implication from the user. An unexperienced programmer may at first, cause a performance loss.
	Extended traditional languages. The start up effort is a bit higher. They also require a different compiler.
	Compilation directives: parallelism is achieved by means of a set of directives that, if ignored, result in a correct sequential program.

Once again, there are complex alternatives combining several of the previous options. The start up effort will depend on their application field (CUDA).

Concurrent programming

Concept: it materializes when the programmer gets involved in the creation, elimination and synchronization of threads, processes or tasks.

Each one of these entities will be executed by a “virtual processor”.

In case the virtual processor is also a real one, it will be parallel programming.

Regardless of where it is developed, the programming environment must provide a series of services:

- A way to express concurrent execution by the explicit declaration of either processes, threads or tasks.
- Communication tools.
- Synchronization mechanisms.

Processes may be:

- Independent: they don’t need neither communication nor synchronization.
- Collaborative: they exchange information to carry out a common job.
- Competitive: they are independent by compete for access to common resources so they will make use of communication and synchronization mechanisms.

Communication may be:

- Shared memory.
- Message passing.

UNIVERSIDAD DE BURGOS

Shared variable synchronization I

Cause: avoid race conditions when accessing critical sections.

Critical section: code area where a shared variable is changed.

Options:

- Busy waiting:
- Semaphores:
- Conditional critical regions:
- Monitors:
- Protected objects:
- Synchronized methods:

UNIVERSIDAD DE BURGOS

Shared variable synchronization II

Busy waiting: before entering a critical section the process checks presence of another process inside. To do so it is necessary to create an indicator and synchronized its use. The complexity of the critical section must offset this effort. It can be carried out by the use of one indicator per process and a “turn” variable. The indicator keeps track of the processes trying to gain access. In case there are more than one, “turn” variable decides. When the process exits the section, “turn” changes.

- This procedure has difficulties to upscale if the number of processes is high.
- Awaiting processes are constantly checking the value of “turn” thus penalizing performance.
- This could be improved through a suspend and resume mechanism.

Semaphores: are integer non negative variables. Two procedures permit their management: wait(decrements its value if >0 otherwise the process waits) & signal(increments its value). Both operations are atomic. Awaiting processes are queued and suspended. Queue management can be performed in various ways; FIFO is the default one.

UNIVERSIDAD
DE BURGOS

Shared variable synchronization III

Conditional critical regions: are sections of code that can only be executed in mutual exclusion. Awaiting processes awake periodically to check access condition which is also accessed in mutual exclusion.

Monitors: modules where critical regions are encapsulated together. It doesn’t provide additional synchronization mechanisms but all variables and procedure calls are accessed in mutual exclusion.

Protected objects, synchronized methods: complementary tools provided by certain programming environment (Ada, Java).

UNIVERSIDAD DE BURGOS

Message passing synchronization

Has a lot to do with what has already been explained about middleware:

- Synchronous systems.
- Asynchronous systems.
- Remote invocation: the sender only resumes operation when it has received a response from the receiver.

UNIVERSIDAD
DE BURGOS

Management of concurrent processes I

Structure: affects process creation:

- The number of processes does not vary.
- It may vary at run time.

Layers:

- Flat model.
- Nested processes.

Granularity:

- Coarse grained: a few processes carry out complex jobs.
- Fine grained: many processes perform simple tasks.

Inicialización:

- Information is given to processes at start up.
- Information is given to processes at run time.

Management of concurrent processes II

Finalization (how processes die):

- When they reach the end of their sequence of statements.
- Suicide: self - finalizing.
- Aborted or killed by other process.
- As a result of an unmanaged error.
- Never (embedded applications).
- When they are no longer necessary (a server process that has no clients left).

Process creation:

- Fork/join: provides dynamic process creation and parameter passing for initialization. Child processes are finalized by their parent. It is a mechanism prone to errors. (C/Pthreads).
- Cobegin: specifies concurrent execution of a sequence of instructions (until a coend if found). Flat model. All processes die when they execute the whole sequence.
- Explicit declaration: the programming language provides the tools to create new processes.

Why concurrent programming?

The model matches real situations: the world is concurrent.

Increases parallel systems performance.

Performance raises even if there is no underlying parallel hardware (autopilot’s example).

UNIVERSIDAD
DE BURGOS

Scheduling

Job management in supercomputers

UNIVERSIDAD DE BURGOS

Introduction

Job scheduling in parallel machines involves decision making about when and how to provide CPU time to processes waiting in a queue.

The impact of scheduling in overall system performance is critical.

There is no algorithmic procedure to obtain the most optimal solution so the alternatives soar.

Multiple factors intervene in this problem so its management is highly complicated.

The scenario where scheduling has to be resolved has changed in the last years. Some architectures have been set aside and some new ones have flooded the market.

Currently nearly all supercomputers are distributed memory machines and an increasing number of them are clusters. In this scenario, memory is no longer considered an independent resource but something that is linked to the processor.

Some relevant terms

Partition:

- the bunch of resources associated to a job. The term derives from the existence of mid-coupled computers.
- Now it is being replaced by other terms such as “chunk” in PBS.

Preemption:

- the ability to suspend ongoing processes to serve higher priority ones.

Alternatives I

Partitioning (resource allocation):

- Static: the resources assigned to an application remain unchanged for the whole execution time.
  - Fixed: the number of processor is determined by the system administrator and no longer modified.
  - Variable: decision is based on user request when submitting the job.
  - Adaptive: partition size is determined by the scheduler at the time of starting the job. It is based on both system load and user requirements.
- Dynamic: the resources allocated may change at run time. Changes will be based on variations of system load and priority levels.

UNIVERSIDAD
DE BURGOS

Alternatives II

Job flexibility:

Rigid jobs:	the number of processors assigned to a job is externally determined and not changed at run time.
Moldable jobs:	the number of processors is determined by the scheduler with some constraints and again is not changed at run time.
Evolving jobs:	the number of resources demanded by the job changes through the different phases of its execution and so does the number of processors allocated.
Malleable jobs:	the scheduler may change the number of processors assigned to a job at run time. It takes into account global system needs to take some processors or assign some more.

Alternatives III

Level of preemption supported:

No preemption:	initiated processes keep all their assigned processors until completion and are never interrupted.
Local preemption:	threads of a job can be stopped but will resume later in the same processor.
Migratable preemption:	suspended threads can resume in a different processor.
Gang scheduling:	all threads of a job are suspended and resumed together with or without migration.

Policies

System administrator may have different goals in mind:

- Minimize wait time for processes in the queue.
- Minimize execution time.
- Maximize throughput.
- Maximize system utilization.

UNIVERSIDAD
DE BURGOS

Final discussion

Dynamic resource allocation tends to be more efficient regardless of the administrator’s aims.

On the other hand it involves a greater complexity since the scheduler has to be aware of the state of execution of all jobs and resource availability during execution. It also has to execute decision making algorithms continuously.

Although wait time in the queue may not be the main goal, a commonly implemented mechanism is the so called “backfilling”. It aims to avail of resources reserved for a waiting process that has not jet obtained all it needs to start execution. Meanwhile smaller processes can make use of theses reserved resources provided they are expected to finish before the awaiting big process has collected all it needs to begin.

UNIVERSIDAD
DE BURGOS

PBS

Queues management and job scheduling

UNIVERSIDAD DE BURGOS

Introduction

PBS (Portable Batch System): workload management system initially developed to manage NASA’s computational resources.

Provides a unified access interface so the users are able to queue their jobs.

Also provides administration tools to grant system managers control over their computational resources.

It is a vendor product but complying with the IEEE 1003.2d standard.

Includes the 3 basic components of every workload management system:

- Queue management: collects the jobs submitted by the users and queues them until the resources they need are available.
- Scheduling: makes job selection and resources allocation policies possible.
- Monitoring: provides tools to track resources utilization in order to help system administrators to optimize system utilization.

UNIVERSIDAD
DE BURGOS

PBS Architecture

Commands

- grant users access to system functionality:
  - User commands: used to submit jobs and track their progress.
  - Administrator commands: to manage the whole system.
  - Operator commands: permit a limited management of the system.

Job server

- includes basic services to issue jobs.

Job executor

- places the jobs into execution.
- returns output to the user when required.
- runs concurrently in every computer expected to execute jobs.

Scheduler

- implements the job selection and resource allocation policies.
- interacts with the executor to know the availability of resources and also with the server to know about the jobs waiting for execution.

UNIVERSIDAD DE BURGOS

Concepts

Node (obsolete)	• a computer system with a single operating system image and a unified virtual memory space.
Vnode (virtual node)	• an abstract object representing a set of resources which form a usable part of a machine. • it is the resource allocation unit.
Host	• machine with its own operating system made up of one or more Vnodes.
Chunk	• a set of resources allocated to a job.
Queue	• container of jobs within a server: • Routing queue: used to move jobs to other queues. • Execution queue: where jobs must be to be eligible for execution.
Walltime	• a job’s execution time.

UNIVERSIDAD
DE BURGOS

Job upload process

User logs in

- usually remotely

A script is used to submit jobs

- qsub<script>

The script includes:

- Shell specification (optional).
- PBS directives: used to request resources (CPUs, memory, licenses) or to specify attributes (walltime).
- Tasks: programs or commands.

The system returns an identifier to allow later operations on the job

- monitoring, deleting, etc.

UNIVERSIDAD
DE BURGOS

Script example

- #!/bin/sh
- #PBS -l walltime=1:00:00
- #PBS -l mem=400mb,ncpus=4
- ./my_application

References

- Modern Operating Systems. Andrew S. Tanenbaum. Pearson Education, 2009.
- http://www.ibm.com/developerworks/linux/library/
- D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, and P. Wong, “Theory and practice in parallel job scheduling,” in IPPS ’97:Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing. London, UK: Springer, 1997, pp. 1–34.
- D. Feitelson, L. Rudolph, and U. Schwiegelshohn, “Parallel jobscheduling–a status report,” Lecture Notes in Computer Science, vol.3277, pp. 1–16, 2005.
- PBS Professional User’s Guide, Altair PBS Professional 10.2
- http://www.youtube.com/watch?v=0G0z8SauSDY#t=55