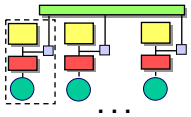
Scheduling

Jesus Labarta

CEPBA-IBM Research Institute

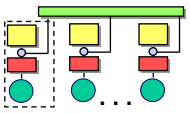
Scheduling

- **Applications submitted to system ➔ Resources x Time**
  - Resources:
    - ✓ Processors
    - ✓ Memory
    - ✓ …



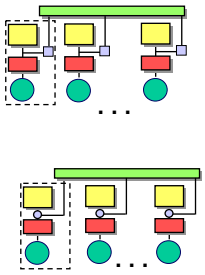
- **Objective**
  - Maximize resource utilization
  - Maximize throughput
  - Minimize response time

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

CEPBA-IBM Research Institute

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
```

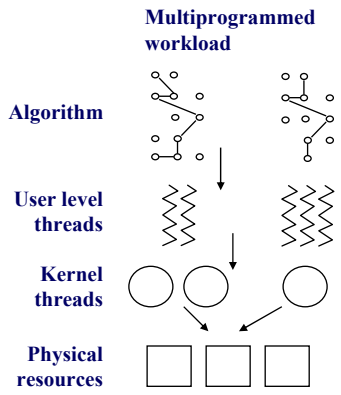
```
C$OMP PARALLEL DO
DO Block=1, #blocks
 Call Compute1(Block)
ENDO
C$OMP END PARALLEL
…
DO iter=1,#iters
   C$OMP PARALLEL DO
   DO Block=1, #blocks
      Call Compute1(Block)
   ENDO
   C$OMP END PARALLEL
END DO
…
C$OMP END PARALLEL
```

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

CEPBA-IBM Research Institute

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



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)

CEPBA-IBM Research Institute

Job scheduling

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

Jesús Labarta, MP, 2002

Job scheduling

- **Basic policies**
  - FIFO
  - SJF
  - Backfilling
- **Requirements**
  - Estimates of job duration
  - Estimates of job performance vs. resources
  - User supplied / prediction

CEPBA-IBM Research Institute

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

CEPBA-IBM Research Institute

Processor scheduling

- **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

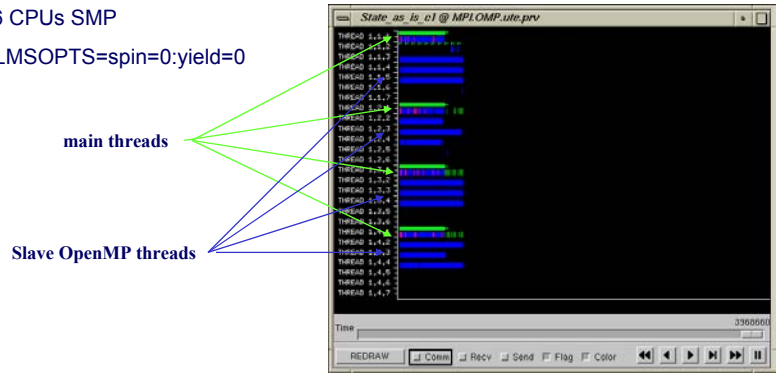
Processor scheduling

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

CEPBA-IBM Research Institute

Motivation example

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

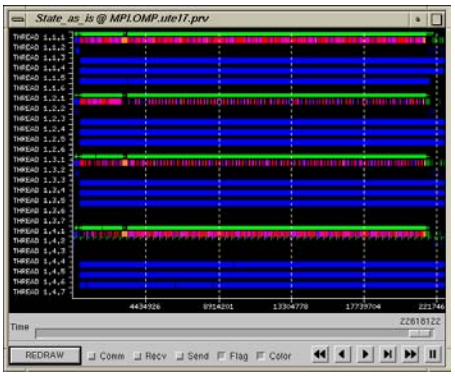


Jesús Labarta, MP, 2002

CEPBA-IBM Research Institute

Motivation example

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



- **+ 1 sequential program**
  - Non instrumented

Same scale as previous slide

CEPBA-IBM Research Institute

Motivation example

- **Quantum == 10 ms**
- **Intrincate precedence and scheduling interfecences**

Very fast communication

Very slow communication

2 threads time sharing CPU

Can not finish because receiver descheduled

Receive called after finising send

Can not finish because sender descheduled

Jesús Labarta, MP, 2002

Motivation example

- **Thread scheduling**
  - XLMSOPTS=spin=1:yield=1

Lots of process migrations of OpenMP slave threads

CPU bursts << Quantum

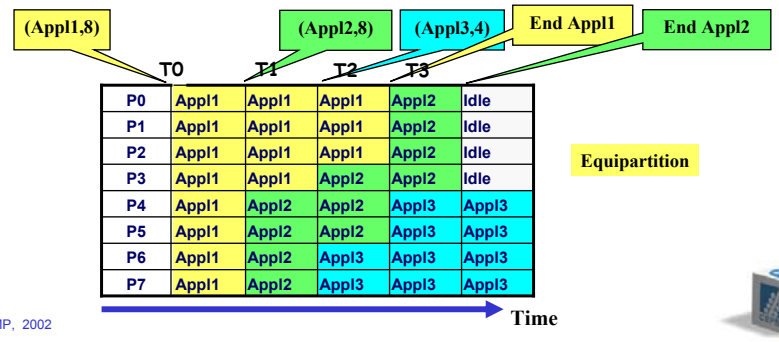
Processor scheduling

- **Space sharing**
  - Partition of processors among applications
    - ✓ Possible fragmentation
  - Example policies
    - ✓ Equipartition
    - ✓ Equal efficiency
- **Gang Scheduling**
  - Coarse grain time sharing

CEPBA-IBM Research Institute

Equipartition

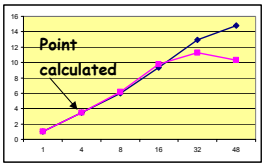
- **Equal allocation among running jobs [McCann93]**
  - ✓ Allocation=P/num_applications
- **Re-allocation at each job arrival and completion**
- **Reasonable approach if job performance not known**



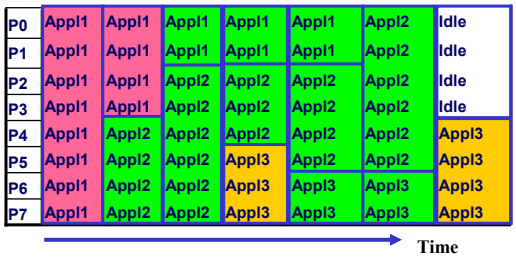
CEPBA-IBM Research Institute

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**



$$Efficiency(p) = \frac{(1+\beta)}{(p+\beta)}$$

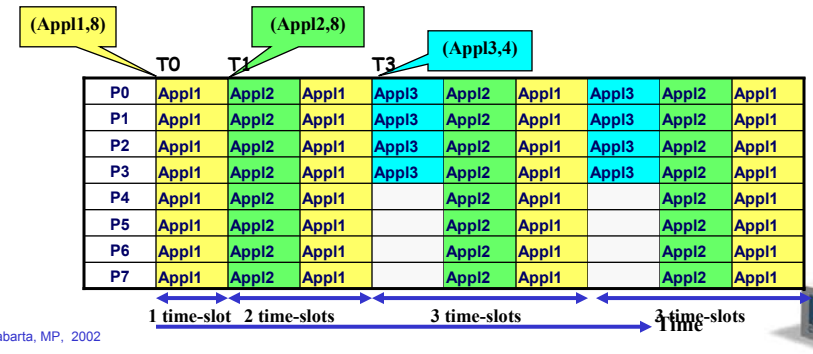


High Efficiency Medium Efficiency Low Efficiency

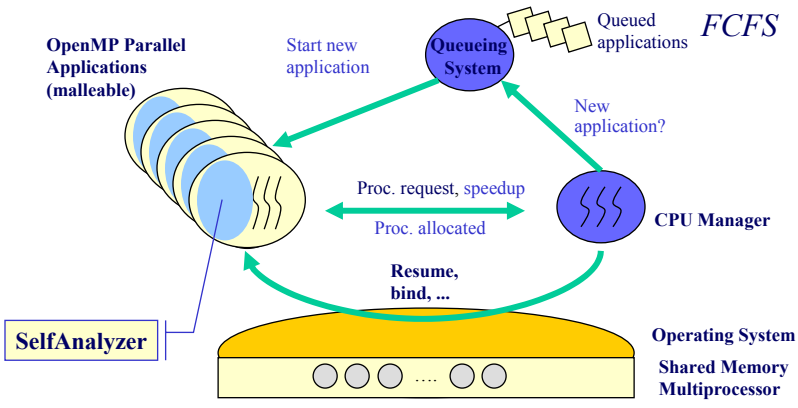
Jesús Labarta, MP, 2002

Gang Scheduling

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



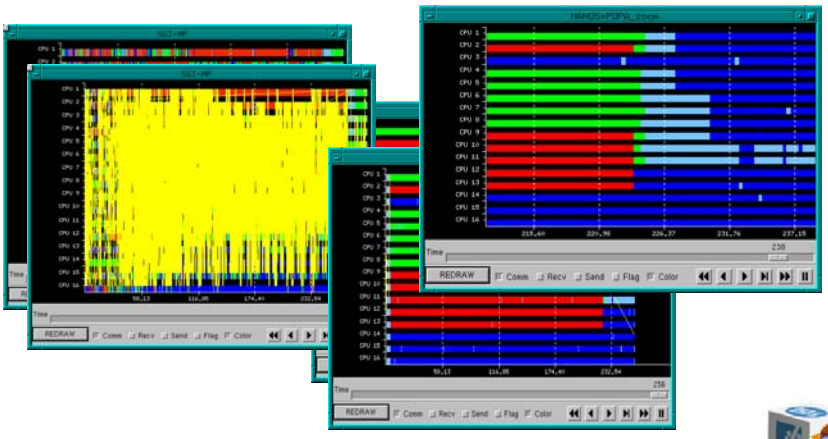
NANOS OS scheduling environment



CEPBA-IBM Research Institute

Native time sharing vs. PDPA

- 3 BTs x 8 processes @ 16 CPUs



Jesús Labarta, MP, 2002

CEPBA-IBM Research Institute

Page placement

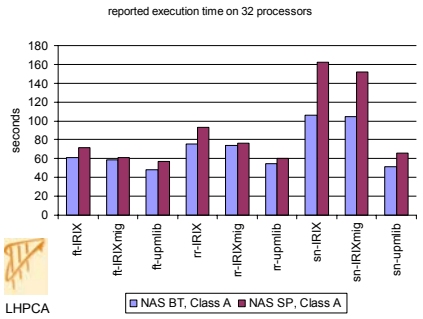
- Mapping of pages to nodes
- UMA
  - Not an issue
- NUMA
  - Round Robin - Random
  - First touch
  - Page migration

Jesús Labarta, MP, 2002

CEPBA-IBM Research Institute

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



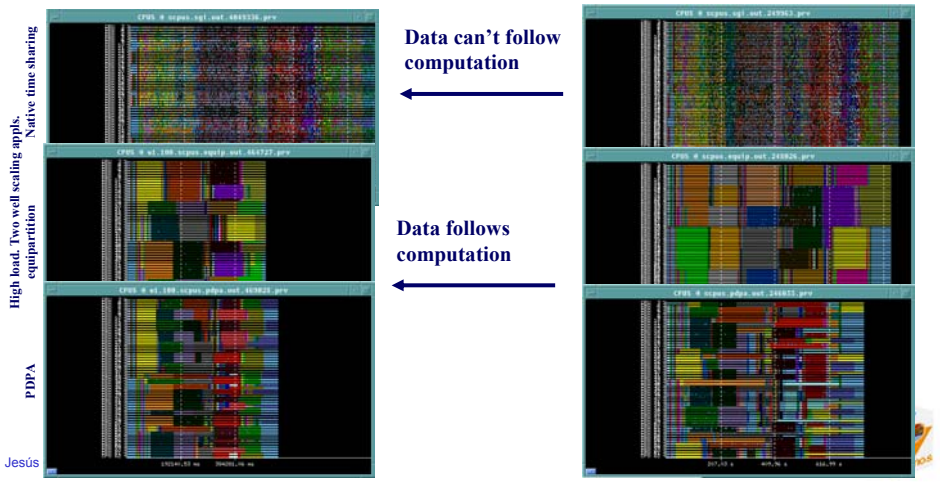
LHPCA

Jesús Labarta, MP, 2002

CEPBA-IBM Research Institute

Page placement & OS scheduling

Native page migration enabled **disabled**



Jesús