1. Motivation

Multiprocessor machines are growing in complexity. Scheduling parallel applications on such machines is a difficult task which requires lots of tuning of scheduling-related kernel code. Operating system developers and system administrators need tools to tune properly the installations under their control.

Operating system general-purpose policies are usually not good enough for parallel applications. For instance, slide 2 presents the effects of a multiprogrammed execution in the execution time of a parallel application. Processor sharing is the main source of overhead due to the operating system scheduling of individual processes. NUMA machines, in addition, provide different performance results depending on the distance between the processors where the application is running and the physical memory where the data is mapped.

The main objective of this work is to present the NANOS Resource Management System (NRMS, slide 3), an environment where parallel applications are scheduled on a multi-processor machine with custom scheduling policies. The improvements in the performance of the system can be analyzed and used as feedback to improve the execution environment.

The NRMS consists of the MP CPU Manager, the MP Interposition Library and the Scpus (show cpus) tool. It has been implemented to run on SGI Origin2000 machines on the context of the NANOS project [http://www.ac.upc.es/nanos].

2. The User-level MP CPU Manager

The MP CPU Manager (slides 3 to 5) is a user-level process which is in charge of controlling the assignment of physical processors to parallel applications running on its environment. It can control all or a subset of the processors in a machine.

During execution, the MP CPU Manager exchanges information with each application through shared memory. Applications inform about the number of resources needed and the CPU Manager answers with the number of processors supplied. This information allows applications to grow or restrict their parallelism when necessary. This means that the environment provided by the CPU Manager favors malleable [1] applications. Non-malleable applications are also supported.

The CPU Manager wakes up every a certain time quantum to distribute the processors. The quantum is set by default to 100 milliseconds.

3. The MP Interposition Library

The communication between the CPU Manager and the applications is done through shared memory. The MP Interposition Library gets control at interesting points during the execution of each application, providing the information to the CPU Manager and getting the number of processors assigned for execution.

The MP Interposition Library is based on DITools [http://www.ac.upc.es/homes/alberts/fpc.html]. It transparently gets control when any application starts execution using the PRELOAD feature of the run-time linker.
4. The Scpus Tool

The Scpus tool is used to give feedback to the user about how is going the real execution. It collects the information regarding processor allocation by sampling the system activity at a given time-step. For each process in the system, the tool determines on which physical processor it is executing. Scpus generates a trace which can be properly viewed using Paraver [http://www.cepba.upc.es/tools/paraver/paraver.htm]. Paraver allows the user to count the number of process migrations that each application has suffered, as long as the global number inside the workload, look at execution times, etc.

5. Scheduling Policies

The MP CPU Manager supports the Cluster scheduling policy (slide 6), which distributes groups of processors among the applications. The default number of processors forming a group is 4. This grouping eases the scheduling and favors the mapping of several processes from the same application to a set of “near” physical processors. Setting the minimum group to 1 processor, the Cluster policy behaves as an equipartition.

The assignment of processors is done through the following steps:

- The scheduling policy checks the requirements of each application in the system and decides how many processors to allocate to each one.
- A mapping algorithm is applied to determine which processors to give to each application. Processor affinity is maintained using the history recorded from the previous quantum.
- The decisions taken are enforced. The CPU Manager supplies the number of processors assigned to each application and for each process allowed to run, it is bound to the physical processor assigned.
- In case some applications remain stopped due to a lack of processors, a round-robin scheme is applied to allow them to execute. The round-robin quantum is set by default to five seconds, large enough to take advantage of the processors caches during each run.

6. Evaluation

The NRMS is being evaluated on an SGI Origin2000 machine (slides 7-9). The experiments show that the NRMS-based environment is obtaining better performance than the native MP Library environment. The reasons are the better coordination between the user and kernel levels and the avoidance of unnecessary migrations of processes among processors.

The experiments show that when a workload is requesting 10 processors and running on top of 8, both the overall execution time of a workload and of each individual application are reduced when running under the control of the MP CPU Manager.

7. Conclusions and Future Work (slide 10)

We have presented the NRMS, a new execution environment based on a user-level CPU Manager that controls the allocation of physical processors to parallel applications. Comparing the execution times of several applications under the NRMS with the results obtained by the native SGI MP Library, we have showed that the CPU Manager provides a smoother execution environment to run efficiently such applications.

Nevertheless, there should be further investigation in process placement and scheduling policies to further improve the exploitation of the target machine. The MP CPU Manager is the starting point for such work.

The NANOS Resource Management System

Xavier Martorell, Julita Corbalán
Nacho Navarro, Jesús Labarta
Dept. d’Arquitectura de Computadors
Universitat Politècnica de Catalunya
Operating Systems Design & Implementation (OSDI 2000)
San Diego, CA, October 23-25, 2000

Motivation

- Workload execution
  - 64-processor SGI Origin2000
  - SPEC95 FP SWIM (5 instances x 16 processors)

- Poor performance due to conflicts in resource sharing
- No coordination between running applications and the kernel
- No global load control

Single vs. workload execution using MPLib

<table>
<thead>
<tr>
<th></th>
<th>MPLib Single instance</th>
<th>MPLib Min. (workload)</th>
<th>MPLib Avg. (workload)</th>
<th>MPLib Max. (workload)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution time</td>
<td>10,51</td>
<td>7,29</td>
<td>8,79</td>
<td>10,51</td>
</tr>
<tr>
<td>Execution on 16 processors</td>
<td>5,51</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Functionality

- CPU Manager functionality
  - Gets applications requests
  - Applies a scheduling policy
  - Computes the number of processors for each application
  - Assigns processors
  - Informs the applications

- Contacting the CPU Manager
  - Interposition mechanism allows to get control before an application executes
  - First contact through a named pipe
  - Dialog through shared memory
  - Application tracking through /proc filesystem

Implementation

- Communication
  - Read/write to shared memory
  - Block/unblock application threads
  - Bind application threads to physical processors
    - Improve memory affinity

- Shared memory contents
  - Requested number of processors
  - Current number of processors
  - Threads identifiers & status

Scheduling Policies

- Cluster
  - Distributes processors in groups of N
  - N-1 results in Equipartition

- Assignment steps
  - Decides how many processors to give to each application
  - Decides which processors to assign
  - Supplies the number of processors and binds them
    - Round-robin

Execution Environment

- NANOS MP CPU Manager
  - Uses the available kernel interface + shared memory
  - Protection guaranteed
  - Malleable applications

- USER-LEVEL scheduler, GLOBAL control

Hardware

Kernel

Application Level

CPU Manager

Shared memory

Thread identifiers & status

Application Level

Kernel Level

User/kernel R/W

CPU

Shared memory

Requested number of processors

Current number of processors

Threads identifiers & status

Communication

- Read/write to shared memory
- Block/unblock application threads
- Bind application threads to physical processors
  - Improve memory affinity

Shared memory contents

- Requested number of processors
- Current number of processors
- Threads identifiers & status
Experiments

Environment
- SGI Origin 2000, OpenMP, MP library
- Workload consisting of three NAS benchmarks (Class A)
  - FT, requesting 4 processors
  - MG, requesting 4 processors
  - CG, requesting 2 processors
- Reasonable system load
  - Using a total of 10 processes
  - Running on 8 cpus
- Local mechanism for load control activated in native environment
- Scpus tool collects traces for later visualization

Results
- Total execution time reduced from 600 to 539 seconds
- CPU Manager performs better than local mechanism for load control
- Further improvements with better mapping policies
- Bad placement for CG (see next slide)
- Open environment, modifications easy introduced

<table>
<thead>
<tr>
<th>Application</th>
<th>Instances</th>
<th>Request</th>
<th>Native SGI</th>
<th>MP CPU Manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>ft.A</td>
<td>10</td>
<td>4</td>
<td>53.2</td>
<td>49.6</td>
</tr>
<tr>
<td>mg.A</td>
<td>10</td>
<td>4</td>
<td>56.9</td>
<td>49.7</td>
</tr>
<tr>
<td>cg.A</td>
<td>10</td>
<td>2</td>
<td>44.0</td>
<td>43.1</td>
</tr>
</tbody>
</table>

MP CPU Manager
- Better coordination
- More consistent execution times
- Native SGI MP Library
- Higher number of process migrations

Conclusions
- MP CPU Manager
  - Kernel scheduling decisions from user level using specific kernel primitives
  - Implementable on top of most current O.S.
- CPU Manager technology helps kernel scheduling
  - Consistent execution times
  - Lower standard deviation
- Highly coordination between user and kernel levels
  - Provides an overall 10% benefit
- More information at:
  - http://www.ac.upc.es/NANOS