Execution Environments for Parallel Applications

Master CANS  2007/2008
Departament d’Arquitectura de Computadors
Universitat Politècnica de Catalunya
Outline

- Supercomputers
- OS abstractions
- Extended OS interfaces
  - IRIX sysmp
  - IRIX memory placement
- Tools for performance analysis
- Example: the CPU Manager
- Linux interfaces
- Conclusions
Why and...
OS Abstractions

Processes, threads, memory, processors...
Process definition

- Resource allocation unit
- Execution support environment, consisting of...
  - Address space
  - Communication channels
  - Execution thread(s)
Processes

- **Address space**
  - Addressable memory, containing code (text), data (data or BSS) and stack
Communication channels
- Data I/O
- Communication with the environment
- Files, sockets, pipes
- Messages and ports
Processes

- **Execution thread**
  - Data structure
  - Program counter + Stack pointer: PC and SP
  - Processor register set
    - Integer
    - Floating point
Process definition

- Execution support environment, consisting of...
- Identified by pid (process identifier)
Processes

- **Process descriptor**
  - Address space
  - File descriptor array
  - Execution thread context
  - Priority: importance
  - Quantum: maximum amount of time before taking a context switch, when there are other threads ready to execute
Process system interface

- Processes in UNIX
  - fork()
  - getpid(), getppid()
  - exec(...)
  - exit(...)
  - wait(...)
Process system interface

- UNIX
  - Process hierarchy

![Diagram showing process hierarchy and PID relationships]

- PID Q
  - getpid() == Q
  - getppid() == P

- PID R
  - getpid() == R
  - getppid() == P

- PID S
  - getpid() == S
  - getppid() == Q
- Data structure allowing the execution
  - System/user-level characteristics
    - Priority: importance
    - Quantum: maximum amount of time before taking a context switch, when there are other threads ready to execute
Threads are useful
- Multiprocessor computers
- Execution of several applications at the same time
- Parallelism among applications

but...
But, can we take advantage of threads inside a parallel application?
Types of parallel applications

- Computing intensive
  - Vectors, matrices, ... simple data types
- Servers, needing support to parallel I/O
  - Data bases
  - Web
 Threads

- **Philosophy**
  - Share all resources that belong to the process
  - Can get access to new resources
  - Can free resources
Threads

- Philosophy
  - Synchronization needed to
    - Access shared data (user / kernel)
    - Access shared resources (kernel)
- **Mixed environment (N:M)**
  - Managed by a user-level library and the kernel
  - Processors $<=$ Kernel threads $<=$ user threads

---

**Thread levels**

- User-level thread descriptor
- Kernel-level thread descriptor
- Physical processor

**Hardware**

**Context switches**

**System**

**User**

**Operating system**

**Apl.**

**Apl.**

**Apl.**

**Apl.**

---

Dept. d'Arquitectura de Computadors

Execution Environments for Parallel Applications
Threads

Advantages

- Can exploit parallelism inside applications
- Lightweight management
  - Efficient context switch
  - No need to flush the entire TLB/page table registers
  - Can use data in the cache
- Save system resources
  - Resource sharing
Kernel-level threads

- Independent abstraction
- Thread identifier
Kernel-level threads

- **Kernel data structures**

  - Process descriptor
    - Address space
    - File descriptor array
    - Root and current directories
    - Thread list
    - Base priority
    - Base quantum

  - Thread descriptors
    - PC+SP
    - PC+SP
    - PC+SP, registers, PPR
    - Priority
    - Quantum
Kernel-level threads

- Operating systems
  - Mach, Tru64 UNIX (Digital UNIX), IBM AIX
  - SGI IRIX 6.5, Linux 2.[456]
  - Solaris, HP-UX (lightweight processes)
  - Windows
  - Chorus
Kernel-level threads

- **Mach, Tru64 UNIX / Digital UNIX**
  - Tasks identified by task_id
    - `task_id = task_self();`
  - Thread creation and management
    - `res = thread_create(task_id, &thread_id)`
    - `res = thread_get_state(thread_id, &context, size)`
    - `// Initialize PC, SP and other registers needed`
    - `res = thread_set_state(thread_id, &context, size)`
    - `res = thread_resume(thread_id)`
Kernel-level threads

- Mach, Tru64 UNIX / Digital UNIX
  - `thread_create (...)

\[ \text{id} = \text{thread\_create} (\text{task\_self} (), \ &\text{th\_id}); \]
Kernel-level threads

- Mach, Tru64 UNIX / Digital UNIX
  - thread_create (...) 

Example
res = thread_create (task_self (), &thid);
if (res!=KERN_SUCCESS) {
    mach_error ("thread_create", res);
    exit (1);
}
res = thread_get_state (thid, &context, sizeof (context));
context.pc = func;
context.sp = malloc (stack_size) + stack_size;
context.a0 = argument;
res = thread_set_state (thid, &context, sizeof (context));
res = thread_resume (thid);
printf ("New thread id %d\n", thid);
Kernel-level threads

- Mach, Tru64 UNIX / Digital UNIX
  - thread_create

**Example**

```c
res = thread_create (task_self (), &thid);
...
res = thread_resume (thid);

void func (int argument)
{
    printf ("thread %d, argument %d\n", thread_self (), argument);
    thread_terminate (thread_self ());
}
```
Shared processes

- Evolution from shared memory
  - As if several processes share memory
Evolution from shared memory
Shared processes

- **OS data structures**
  - naturally shared

Process descriptor

- PC+SP, registers, FPR
- Priority
- Quantum
- Address space
- File descriptors
- FS information
  ...

Shared resources

- Memory regions
- Open channels
- Current/root directories

Process descriptor

- PC+SP, registers, FPR
- Priority
- Quantum
- Address space
- File descriptors
- FS information
  ...
Comparison with kernel threads

- **OS data structures**
  - Lots of changes
    - Kernel structures, linked lists, ready queue...

**Process descriptor**
- Address space
- File descriptor array
- Root and current directories
- Thread list
- Base priority
- Base quantum

**Thread descriptors**
- PC+SP, registers, FPR
- Priority
- Quantum

**Comparison with kernel threads**
Shared processes

- ... are in
  - Linux (clone)
  - IRIX (sproc)
- ... but they are in “danger of extinction”
- **Linux clone**
  - pid = clone (func, usp, flags, arg)

Diagram:
- Shared address space
- pid0
- Thread
- pid1
- pid2
- pid3
- clone (...)
Linux clone
- pid = clone (func, usp, flags, arg)

Ejemplo

```c
pid = clone (func, usp, flags, argument);
if (pid<0) {
    perror ("clone");
    exit (1);
}
printf ("New clone pid %d\n", pid);
res = wait (&status);
... 
void func (void * arg)
{
    int argument = (int) arg;
    printf ("clone %d, arg %d\n", getpid (), argument);
    exit (0);
}
```
Shared processes

- **Clone system-call flags**
  - CLONE_VM, shared address space
  - CLONE_FILES, shared file descriptors
  - CLONE_FS, shared root and current directories
  - CLONE_SIGHAND, shared signal programming
  - ... other
  - Signal to be sent to the parent process on child termination

- **fork() is a special case of clone!**
#include <stdio.h>
#include <signal.h>
#include <sched.h>

#define STACK_SIZE 8192
int val = 44;
int newclone (void * arg);

void main ()
{
    int status;
    int pid;
    char * ustack = (char *) malloc (STACK_SIZE);
    if (ustack==NULL) {
        perror ("malloc");
        exit (1);
    }
    pid = clone (newclone, ustack+STACK_SIZE,
        CLONE_VM | CLONE_FILES | CLONE_FS |
        CLONE_SIGHAND | SIGCHLD, 55);
    if (pid<0) {
        perror ("clone");
        exit (1);
    }
    printf ("Clone created pid %d\n", pid);
    pid = wait (&status);
    if (pid<0) {
        perror ("wait");
        exit (1);
    }
    printf ("Clone %d finished (status %x)\n", pid, status);
    printf ("val %d\n", val);
    free (ustack);
}

int newclone (void * argument)
{
    int arg = (int) argument;
    printf ("Newclone %d, argument %d\n", getpid (), arg);
    val = argument;
    return 0; /* exit (0); */
}
UNIX process interface

- UNIX System calls for process management
  - pause ()
  - signal/kill SIGSTOP, SIGCONT
  - nice

- Reduced functionality
  - No ability to manage physical resources (processors, memory...)

- Not versatile enough
  - It can not be used to modify the characteristics of other processes
Extended process interface

- **IRIX**
  - blockproc (pid)
  - unblockproc (pid)

  - blockprocall (pid)
  - unblockprocall (pid)

  - setblockproccnt (pid, count)
  - setblockproccntall (pid, count)
    - count < 0 means process blocked
  - prctl (PR_ISBLOCKED, pid);
Extended process interface

- **IRIX**
  - **prctl**
    - **PR_MAXPROCS**
    - **PR_MAXPPROCS**
    - **PR_ISBLOCKED**
    - **PR_GET/SETSTACKSIZE** (also setrlimit)
    - **PR_UNBLKONEXEC**
      - Race-free process creation
    - **PR_SETEXITSIG**
      - All share group is signaled
    - **PR_RESIDENT**
      - No swap-out, please
      - In addition mpin() can lock the pages in memory
Extended process interface

- **IRIX**
  - **prctl**
    - **PR_ATTACHADDR, pid, addr**
      - Only for share groups
    - **PR_ATTACHADDRPERM**
      - Only for share groups
      - Local address, pid, remote address, newprotection, shared/private
    - **PR_GETSHMHMASK**
      - Which resource are shared
    - **PR_GETNSHARE**
      - Number of processes in the share group
    - **PR_COREPID**
      - Generate core.pid for the share group
    - **PR_THREAD_CTL**
      - Block/unblock/bind/unbind for kernel threads
Extended process interface

- **IRIX**
  - **sysmp**
    - **MP_SCHED, schedctl**
      - AFFINITY_ON
      - RENICE
      - SLICE
      - SETHINTS
      - SCHEDMODE
        - SGS_FREE
        - SGS_SINGLE
        - SGS_GANG
    - **MP_MUSTRUN**
    - **MP_MUSTRUN_PID**
    - **MP_RUNANYWHERE, _PID**
Processor interface

- IRIX, control on processes is not enough
  - sysmp
    - MP_RESTRICT
    - MP_ISOLATE
    - MP_NONPREEMPTIVE (no clock scheduler)
    - MP_SAGET, system activity
      - CPU_IDLE
      - USER
      - KERNEL
      - WAIT (IO, SWAP, PIO, GFXC, GFXF)
      - SXBRK
      - INTR
Machine description interface

- IRIX, control on processors is not enough
  - sysmp
    - MP_NPROCS, MP_NAPROCS
    - MP_NUMNODES
    - MP_NUMA_GETCPUNODEMAP
    - MP_NUMA_GETDISTMATRIX
    - MP_PGSIZE, getpagesize()
    - MP_STAT
      - cpuid
      - flags (MASTER, CLOCK, ENABLED, ISOLATED, NONPREEMPTIVE, NOINTR...)
  -
Each process may map a kernel data structure to be informed about its own scheduling:
- pid
- hint, SGS_FREE, SGS_SINGLE, SGS_GANG
- processor type
- thread id
- Last/current cpu
Need for Performance Analysis

- Complexity grows in all the areas
  - Computers
  - Processors
  - Operating System
  - Run-time libraries
  - Applications

- We have a problem...
Tools for Performance Analysis

Performance Analysis Portable Interface
Performance analysis

Why such a loop is performing so bad???

 !$OMP PARALLEL DO
do i = 1, N
do j = 1, N
 A(i,j) = 0.0

Is it a matter of TLB misses?
Is Primary data cache misses?
External interventions???
False sharing?
Performance analysis

- Can we obtain statistics about performance?
- Resource utilization
  - Caches
  - Memory
  - FP instructions
Performance analysis

- Tracing is necessary to figure out about performance
- Need for low interference
- Two views
  - Internal to the application
  - External from the operating system
    - When several applications executed at a time
Architectural support

- **Performance event counters**
  - User
  - Kernel
  - Exception

- **Limitations**
  - Small number of registers to count on
    - Multiplex
  - Small register size (32bits, R10K)
    - Overhead
  - Hardware dependent
    - Processors from the same family have different counters
Architectural support

- **Interesting performance events**
  - Cycles
  - Issued, graduated instructions
  - Issued, graduated loads/stores
  - Level N cache misses
  - TLB hits / misses
  - ALU/FPU progress cycles
  - Number of specific FP instructions issued, graduated
  - External intervention hits
  - External invalidations
  - and many more...

- **Deriving other metrics**
  - Memory bandwidth
  - MFlops
  - IPC
OS support

- None (...)
  - Currently, Linux
  - But there are patches providing access to cpu counters
    - perfctr

- Reading/writing global counters
  - Some support libraries

- Performance counters on thread/process context
  - IRIX, AIX, Solaris...
OS support

- **Process/thread information**
  - /proc
  - system calls

- **Memory/swap usage**

- **[Processor information]**
  - [Which process is currently executing?]
  - How many time is spent in OS/exception code?
OS support

☐ Availability of information related to processors
  ☐ Time spent in... [sysmp (MP_SAGET, SINFO_CPU, &data)]
    ☐ User level
    ☐ System level
    ☐ Waiting (IO, SWAP, PhysIO, Graphics...) [SGI]
    ☐ Memory management
    ☐ Interrupt
    ☐ Idle
  ☐ [Process currently executed]
    ☐ Not usually available :-(
    ☐ Process to processor information usually reported
RTL support

- **Counting**
  - User-defined event-sets
    - Groups of counters got as a unit of information
    - Move counters as a bulk across the OS interface
    - Specialize use of counters and relate different ones
  - Multiplex
  - Overflow detection
    - User-programmed routines
RTL support

- **Multiplex**
  - Limited number of events counted at a time
  - Software can switch events counted
    - Resolution depends on interval timers
  - Overhead in the implementation

- **Inaccuracy**
  - Short high performance routines hidden
User-defined handlers

- Activated on an event exceeding a specific threshold
- Usually through OS support
  - Otherwise an alarm handler can help
- Implemented through signals... efficient???
RTL support

- Getting/extracting the information
  - From the kernel
    - Specific system call needed
    - Reading /proc
  - Directly from the processor
    - Such machine language instructions are usually privileged
    - Pentium architecture allows them to be read from user-level
      - Setting a bit in the processor (CR0?)
  - Always will read own counters
Support to the end-users

- **Profiling tools**
  - Traces are analyzed/visualized by end-user utilities
  - Prof generates histograms
    - Number of events per subroutine/program line
  - PARAVER
  - Vampir
Goal

- Specify a standard API for accessing hardware performance counters

Defines

- Standard set of hardware events
- Library interface
Architecture

- PAPI low level
  - Multiplex counters
  - Overflow management
  - Timer interrupt

- PAPI machine dependent substrate

- Kernel extension

- Operating System

- Performance Counter Hardware

- PAPI high level
How to use

Performance Analysis Tools
Feedback-Directed Compilers
Adaptive Run-Time Libraries
Application measurement and timing

PAPI low level
- Multiplex counters
- Overflow management
- Timer interrupt

PAPI high level
Processor / OS supported

- POWER3, AIX5
- POWER4, AIX5
- CRAY T3E, Unicos
- AMD Athlon, Pentium III, Linux [patched]
- Pentium III, Windows
- Itanium I & II, Linux [patched]
- UltraSparc I, II & III, Solaris 2.8
- MIPS R10K, R12K... IRIX
- Alpha, Tru64 UNIX
- Alpha, Linux [patched]
- BG/L, Linux/CNK
Overview of the interface

- Set of hardware independent events
  - Not available in all hardware
  - Check list!
- Low level API
  - Counter management and acquisition
- High level API
  - Easier counter acquisition
Low-level PAPI

PAPI_create_eventset (eventSet)
- Creates an eventSet. Not thread-safe

PAPI_destroy_event_set (eventSet)

PAPI_add_event (eventSet, event)
PAPI_add_events (eventSet, events, number)
PAPI_cleanup_eventset (eventSet)
PAPI_rem_event (eventset, event)
PAPI_rem_events (eventSet, events, number)
- EventSet management
Low-level PAPI

- **PAPI_get_hardware_info (hw_info)**
- **PAPI_get_executable_info (exe_info)**
- **PAPI_get_opt (option, value)**
  - Access to the environment
- **PAPI_query_event (eventCode)**
  - Checks whether the hardware support such event
- **PAPI_set_domain (domain)**
  - Execution domain to count
    - User, kernel...
- **PAPI_set_granularity (granularity)**
  - thread/process/process group/current cpu/all cpus
Low-level PAPI

- **PAPI_get_overflow_address (context)**
  - Gets overflow PC

- **PAPI_overflow (eventSet, eventCode, threshold, flags, handler)**
  - Sets the overflow handler for the indicated event in eventSet

- **PAPI_set_multiplex (eventSet)**
  - Allows multiplexion

- **PAPI_get_real_cyc (long long)**

- **PAPI_get_real_usec (long long)**
  - Time control
Low-level PAPI

PAPI\_start (eventSet)
- Starts counting...not if in conflict with another eventSet

PAPI\_stop (eventSet, values)
- Stops the eventSet and reads values

PAPI\_accum (eventSet, values)
- Accumulates current counters in eventSet to values

PAPI\_read (eventSet, values)
- Copies current counters in eventSet to values

PAPI\_write (eventSet, values)
- Copies the values to the current counters in eventSet
High-level PAPI

- PAPI_start_counters (events, number)
- PAPI_read_counters (values, number)
- PAPI_accum_counters (values, number)
- PAPI_stop_counters (values, number)
- PAPI_flops (...)

UPC
Performance visualization

- Merging data from...
  - tracing applications
  - hardware event counters

- Example: visualization with Paraver
- **What can we get?**
  - Thread status
  - Performance counter values
  - Source code line