Disco : Running commodity operating systems on scalable multiprocessors
Edouard et al.

Presented by Jonathan Walpole

(based on a slide set from Vidhya Sivasankaran)

Outline

- Goal
- Problems & Solution
- Virtual Machine Monitors(VMM)
- Disco architecture
- Disco implementation
- Experimental results
- Conclusion

Goal

- Develop OS to run efficiently on new shared memory multiprocessor hardware with minimal effort
- Make extensive use of existing OS and application code base
- Utilize Virtual Machine Monitors (VMM) – VM runs multiple copies of OS on a scalable multiprocessor

Problems

- The system software for scalable multi-processor machines lags far behind the hardware
- Extensive custom modifications to OS needed to support new scalable machines
- Modification is implementation intensive and has reliability issues
- Backwards compatibility is important

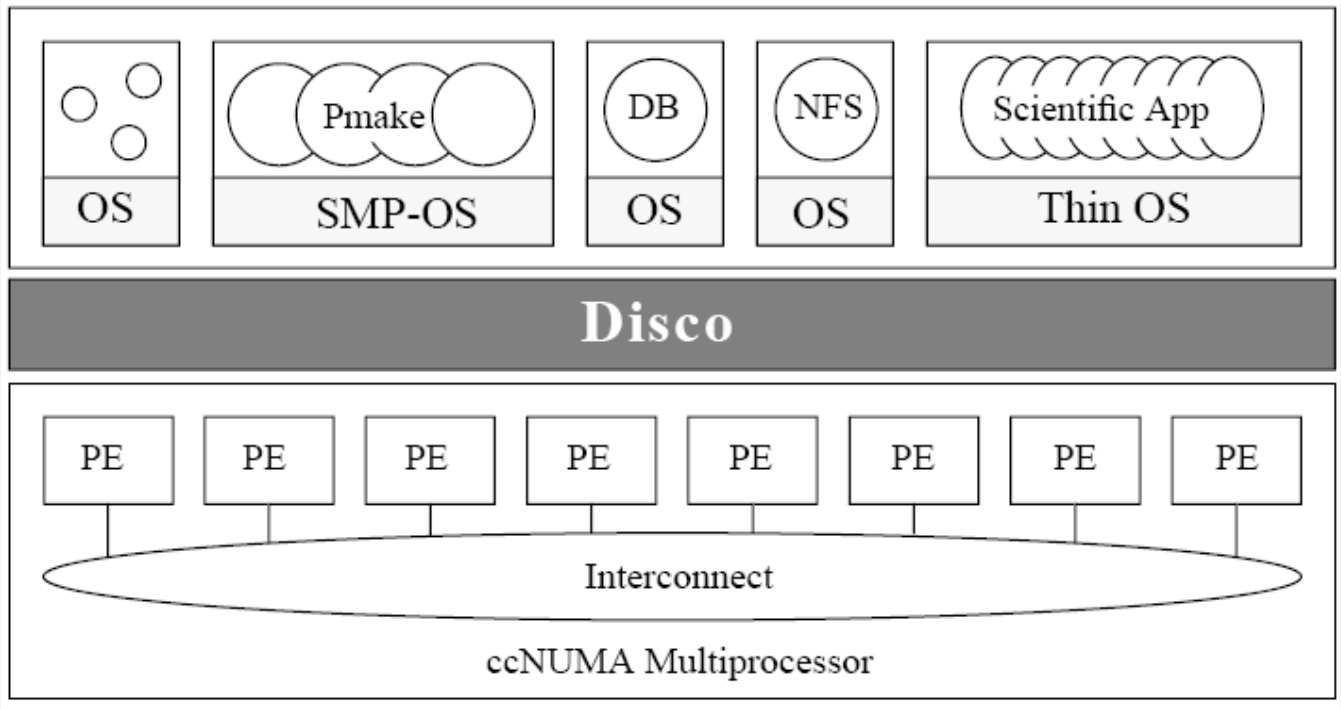
Solution

- Insert a Virtual Machine Momitor (VMM) between existing OS code and new hardware
- The new VMM described in this paper is called Disco

Virtual Machine Monitors

- VMM is a software layer
- It *virtualizes* all system resources in order to export a conventional hardware interface to OS code
  - Multiple OS instances can run concurrently on the same physical hardware
  - They think they each have their own real machine, but it is virtual
  - They are protected from each other
  - They can communicate with each other using Internet protocols
- But doesn't this approach imply a lot of overhead?
- And what's the benefit?

Disco Architecture.



Advantages

- Scalability?
  - Allows use of scalable hardware by multiple non-scalable OS's
  - A large shared memory multiprocessor looks like a network of smaller machines
- Fault containment
  - Even though they are on the same physical hardware they are protected from each other
- Avoid NUMAness
  - Non-uniformity of memory can be hidden by the VMM
- Flexibility
  - Can run existing OS code unmodified or new specialized OS code

Challenges

- Overhead
  - Isn’t there a lot of replication?
- Resource management
  - Does the underlying VMM have enough information to make good decisions?
- Communication and sharing
  - VMMs used to be independent
  - Now they can use Internet protocols to talk to each other

Disco Implementation

- The VMM is a concurrent shared memory program
- Attention given to Numa and cache-aware data structures
- Code segment of disco is replicated in local memory of each processor
- Communication (via Internet protocols) actually uses shared memory

Virtual CPU

- Disco emulates the execution of each virtual CPU by using direct execution on the real CPU.
  - Disco sets the registers of the real CPU to those of the virtual CPU and jumps to the current PC
  - The state for each virtual CPU is kept in a data structure (like a process control block)
- Each virtual CPU of Disco provides the abstraction of a MIPS R10000 processor

Virtual CPU (contd..)

- Disco runs in privileged mode
- All OS code runs in supervisor mode which does not allow execution of privileged instructions
  - So how can OS code execute privileged instructions?
- Attempts to execute privileged instructions trap to Disco. Disco executes them on behalf of the OS, limiting access to that OS's VM resources

Virtual Physical memory

- Each VM has its own physical pages, but they are not necessarily contiguous
- Each OS *thinks* it has access to contiguous physical memory starting at address 0
- Disco keeps track of the mapping between real memory addresses and each OS's physical memory addresses using a pmap structure
- When OS tries to insert a virtual-physical address mapping in the TLB, Disco intercepts this (because updating the TLB is privileged) and inserts the real memory address in place of the physical address.
- The TLB does the actual address translation at normal speed

Continued..

- In OSs designed for the MIPS processor, kernel memory references bypass the TLB and directly access memory
  - Kernel runs in “physical addressing mode”
  - This would violate VM protection
  - Need to relink OS code and data to run in virtual addressing mode
- Workload execution on top of Disco suffers from increased TLB misses
  - switching VMs requires flushing TLB
  - Switching from application to guest OS requires flushing TLB
  - A large software TLB can lessen the performance impact

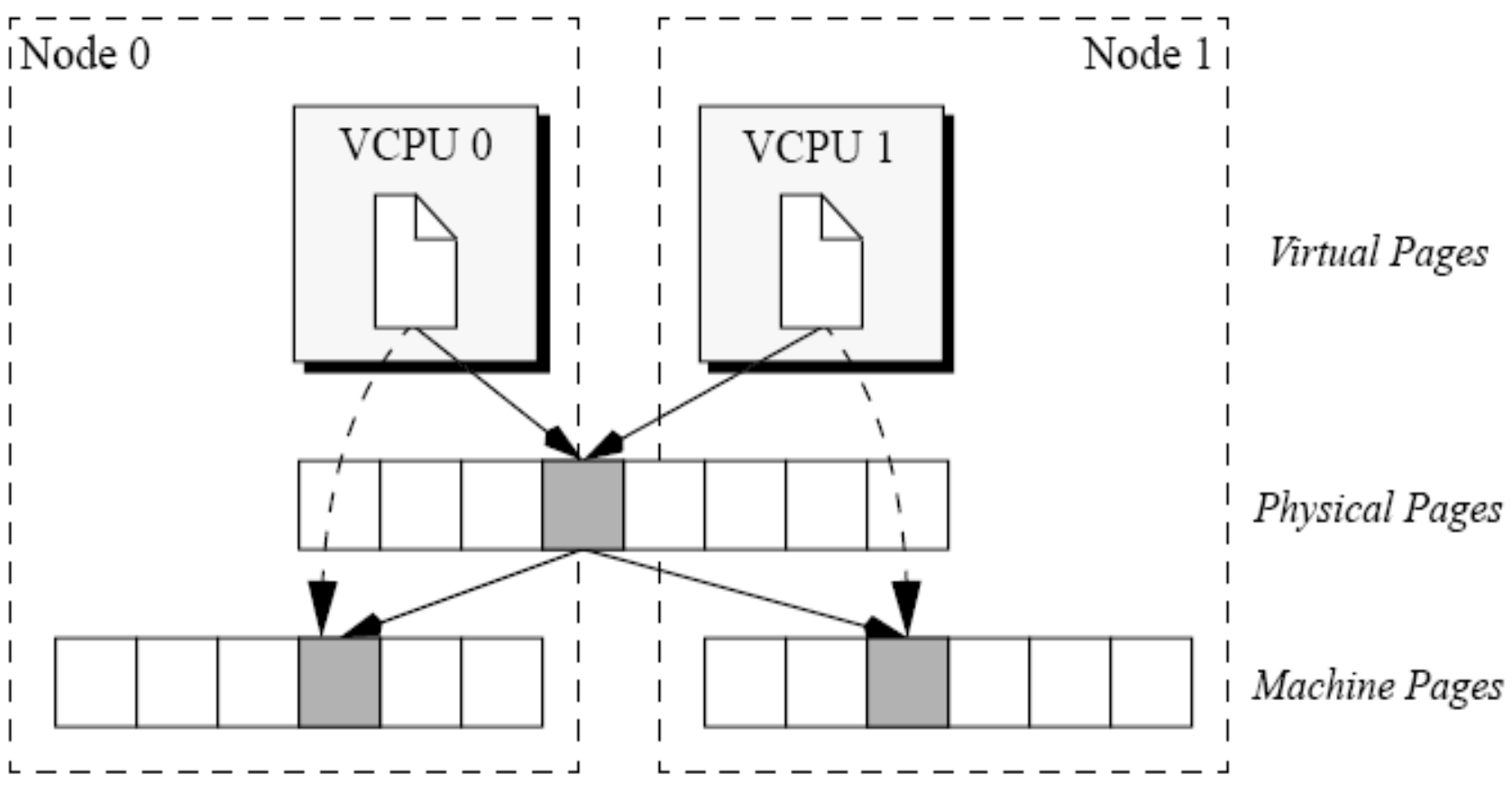
NUMA Memory Management

- On a cache coherent NUMA machine the system will work correctly regardless of where data is placed in memory.
  - However, ideally, cache misses should be satisfied from local memory to reduce memory access latency
  - Disco implements dynamic page replication and migration to build the illusion of a UMA machine on a NUMA machine
- Page Migration
  - Heavily accessed pages by one node are migrated to that node
  - Disco transparently changes the physical-machine address mapping
  - Invalidates the TLB entry mapping the old machine page then copies the data to the new, local page

Continued..

- Page Replication
  - For pages that are frequently accessed using reads by multiple nodes
  - Downgrade the TLB entry of the machine page to read-only and then copy the page to local node and update its TLB entry
  - Disco maintains a data structure, called memmap, with entries for each real machine memory page.

Page replication



Disco uses physical to machine mapping to replicate the pages.Virtual page from both cpu of same virtual machine map the same physical page of their virtual machine.Disco transparently maps each virtual page to machine page replica that is located local to the node.

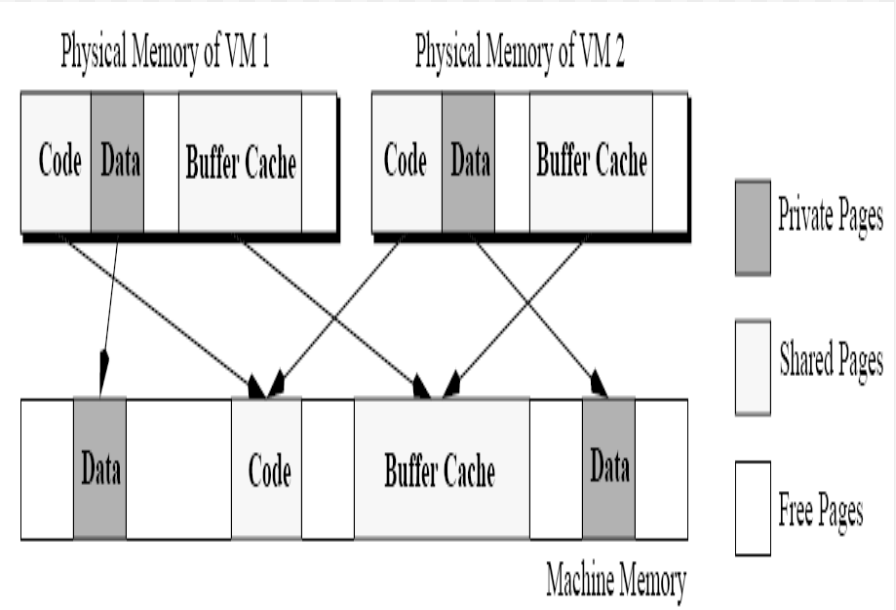
Virtual I/O devices

- To virtualize I/O devices Disco intercepts all device accesses from a virtual machine and passes them to physical devices
  - Rather than interposing on traps, Disco requires device drivers to use a special interface that calls Disco

- Naïve (expensive, but transparent) approach:

1. VM executes instruction to access I/O
2. Trap generated by CPU (based on memory or privilege protection) transfers control to VMM.
3. VMM emulates I/O instruction, saving information about where this came from

Copy on write disk

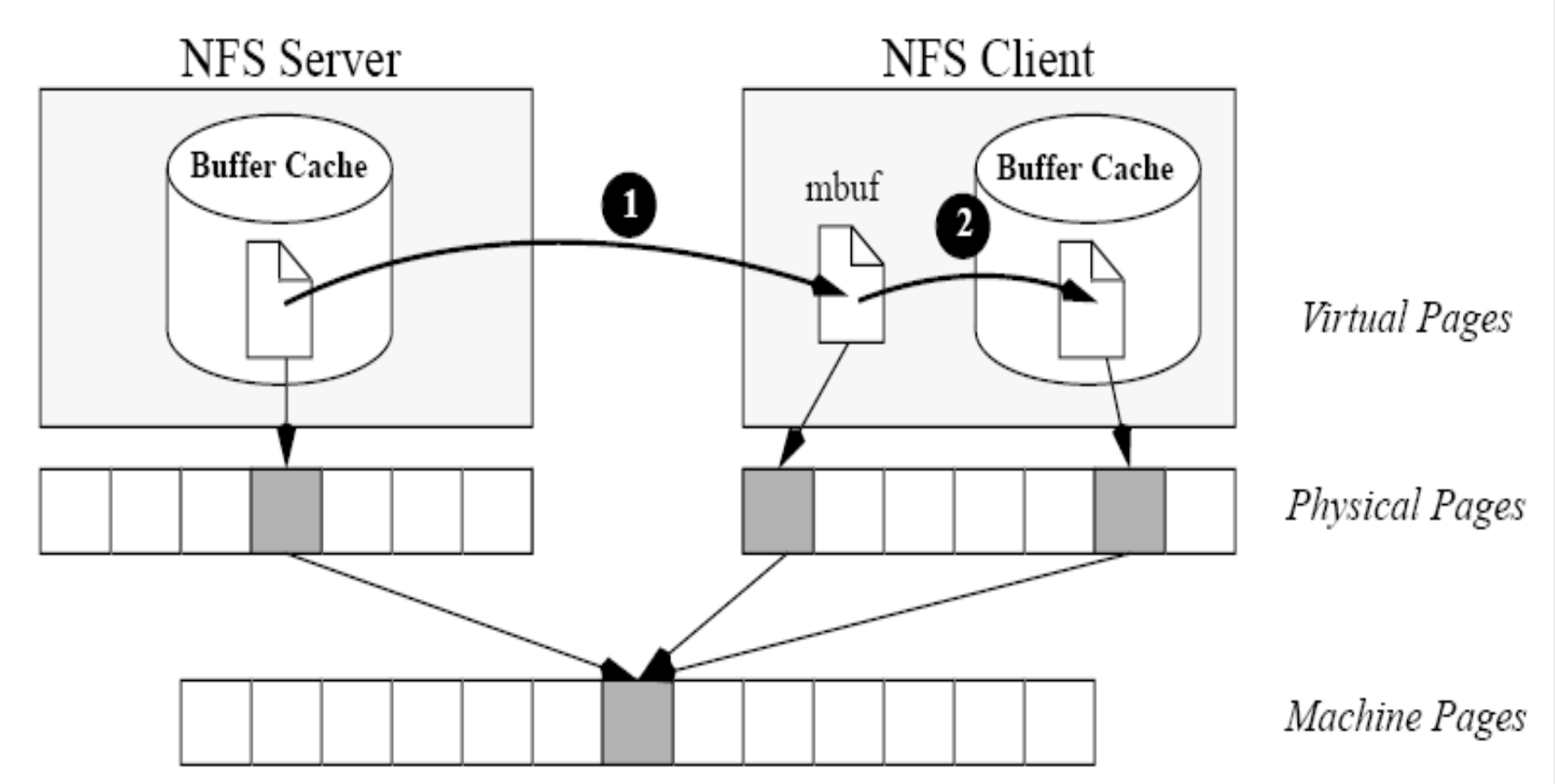


- Disk reads can be serviced by monitor, if request size is multiple of machine page size, then monitor has to remap the machine pages into VM physical memory.

- Pages are read only and an attempt to modify will generate copy on write fault handled by monitor.

Read only pages are are brought in from disk can be transparently shared between virtual machines.This creates global buffer shared across virtual machine and helps to reduce memory foot prints.

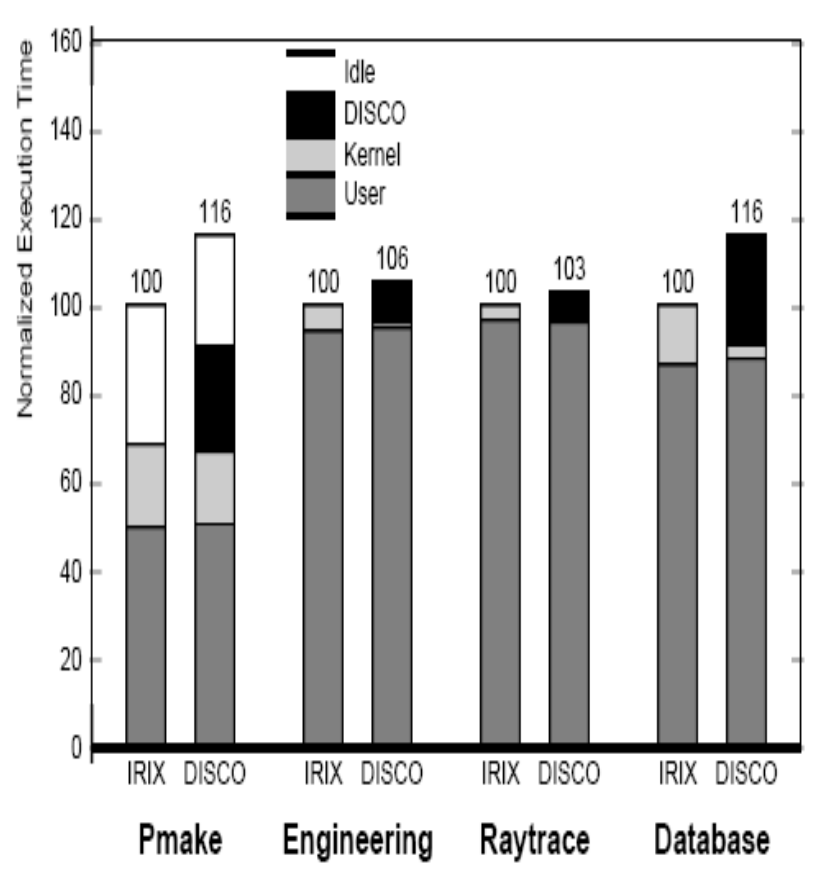
Virtual N/W interface



1)monitors n/w device remap data page from source machine address to destination machine address.

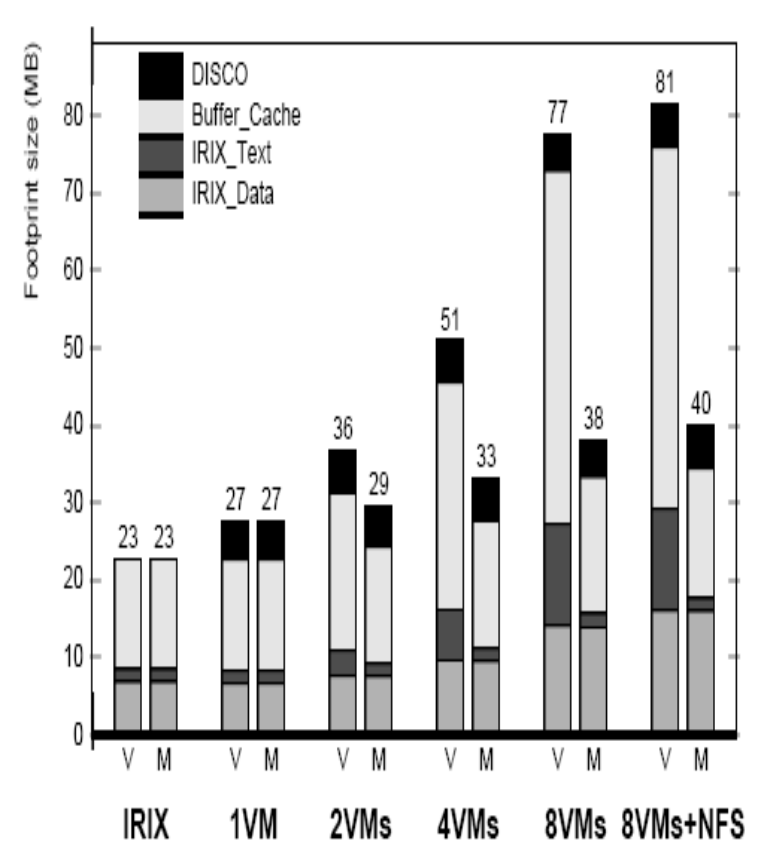
2)monitor remap the data page from drivers mbuf to client buffer cache.

Execution Overhead



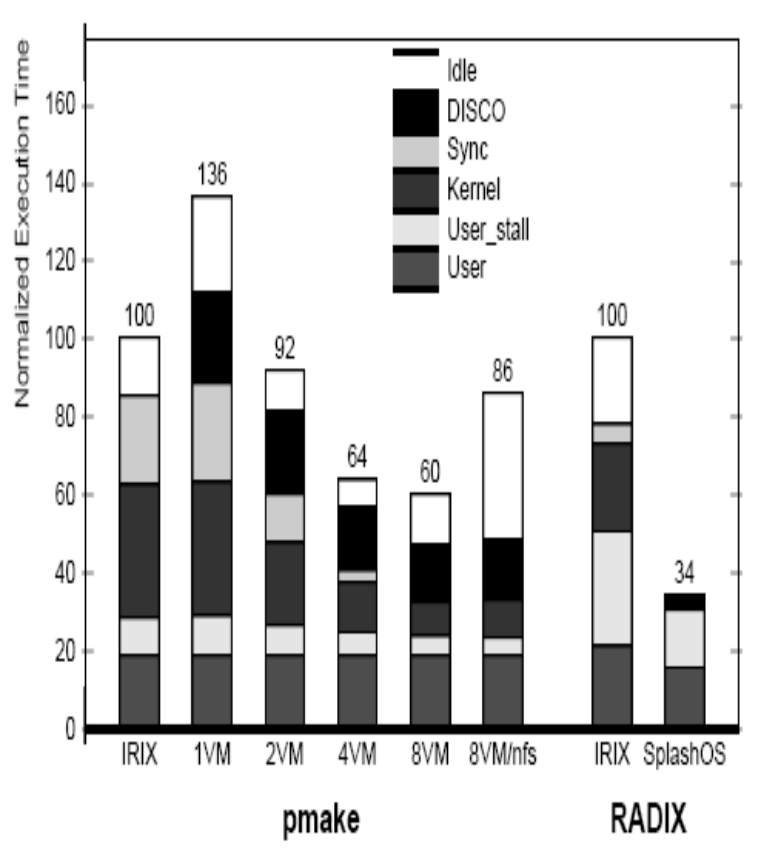
- Experimented on a uni-processor, once running IRIX directly on the h/w and once using disco running IRIX in a single virtual machine
- Overhead of virtualization ranges from 3% - 16%.

Memory overhead



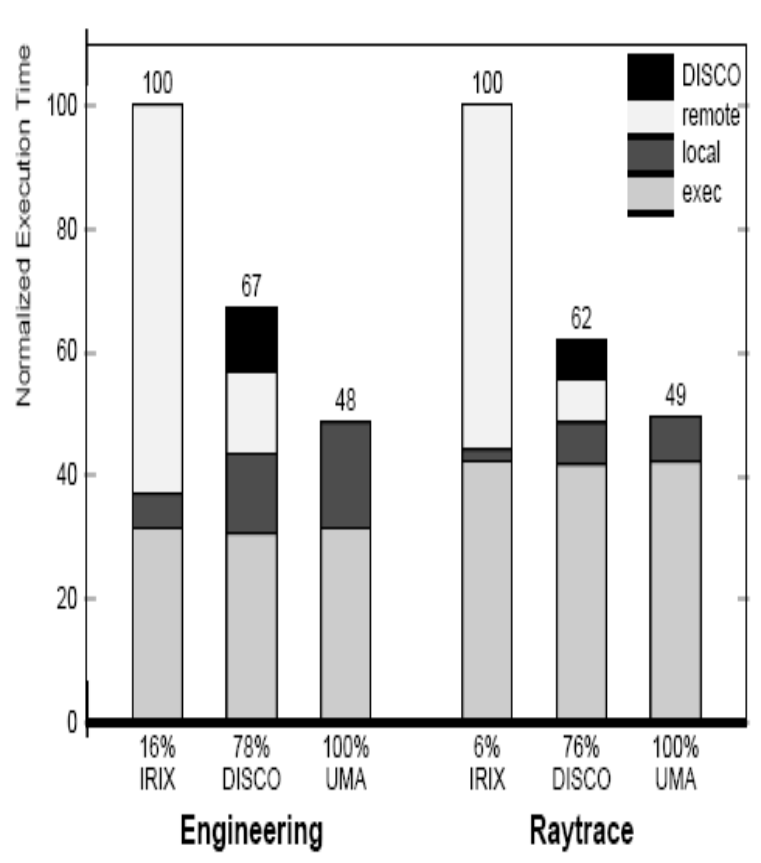
- Ran single workload of eight different instances of pmake with six different system configurations
- Effective sharing of kernel text and buffer cache limits the memory overheads of multiple VM's

Scalability



- Ran pmake workload under six configurations
- IRIX Suffers from high synchronization overheads
- Using a single VM has a high overhead. When increased to 8 VM's execution time reduced to 60%

NUMA



- Performance of UMA machine determines the lower bound for the execution time of NUMA machine
- Achieves significant performance improvement by enhancing the memory locality.

Conclusion

- Developed system software for scalable shared memory multiprocessors without massive development efforts
  - ... and soon after made barrow loads of $$$!
- Results show that overhead of virtualization is modest
- Provides solution for NUMA management