Announcements

- Programming Assignment #1 was handed out
  – PVM Programming card is on the class web page
- OpenMP paper is available from Dept. Library
- Photos are now on the class Web Page
  – See Dr. Hollingsworth for the username/password
- Reading
  – Today 4.1 & PVM paper
  – Thursday MPI & OpenMP

copyright 1999 Jeffrey K. Hollingsworth

Synchronization

- Semaphores
  - Traditional uni-processor synchronization
  - provide blocking wait
  - generally require kernel support
    - implies a kernel trap for each operation (expensive)
    - can involve a full context switch
      - very expensive (1000's of instructions)
- Test-and-set
  - Traditional uni-processor synchronization
  - use busy wait
  - very little kernel support
    - just provide a shared region of memory

copyright 1999 Jeffrey K. Hollingsworth

Synchronization (cont.)

- Spin-locks
  - really just an abstraction of test-and-set used for mutual exclusion
  - still use busy wait
- Hybrid spin and block
  - spinning is great if the delay is “short”
  - blocking is better if the delay is “long”
  - hybrid is spin for a while
    - if get the lock continue
    - if time-out reached, then delay
  - Key parameter is the cut over between spin and block

copyright 1999 Jeffrey K. Hollingsworth

Hybrid Spin Algorithms

- For Additional Information on this topic:
  - A. R. Karlin, K. Li, M. S. Manasse, and S. Owicki, Empirical Studies of Competitive Spinning for a Shared-Memory Multiprocessor, in 13th ACM Symposium on Operating System Principals, 1991.
  - T. E. Anderson, "The Performance Implications of Spin-Waiting Alternatives for Shared-Memory Multiprocessors", ICPP, 1989, pp. II:170-174.

copyright 1999 Jeffrey K. Hollingsworth

Barriers

- a set of processes k all leave the synchronization region at once
  - “at the same time” is hard in a parallel system
  - sufficient that no process leaves until all process arrive
- can be expressed as a busy wait on shared memory
  - hardware support: fetch-and-add instruction
  - built from test-and-set instruction
    - need to provide atomic update of the counter
  - creates a memory hot spot for the count variable
    - can design the memory system to avoid this
      - use a cache update protocol
      - processors spin on the cached value

copyright 1999 Jeffrey K. Hollingsworth

Barriers (cont.)

- can be built as a series of messages
  - all processes send to a barrier coordinator
  - use a tree to reduce the work of the coordinator
    - each process combines $\log_m n$ messages
    - total messages is still O(n)
    - need to scale network too
- are a instance of a general operation called a reduction
  - a commutative operator
  - each process contributes a value

copyright 1999 Jeffrey K. Hollingsworth

Synchronization (cont.)

- Rendezvous
  - defined as part of the language Ada
  - two zero buffered send/receive pair
  - each process blocks until the other arrives
- RPC
  - tries to simulate a traditional procedure call interface
  - sort of a language independent rendezvous
- Futures
  - promise for data to be delivered in soon
  - functions can return immediately a future
  - program blocks if the data has not yet arrived and it is used
  - sort of like a dataflow model, but at the language level

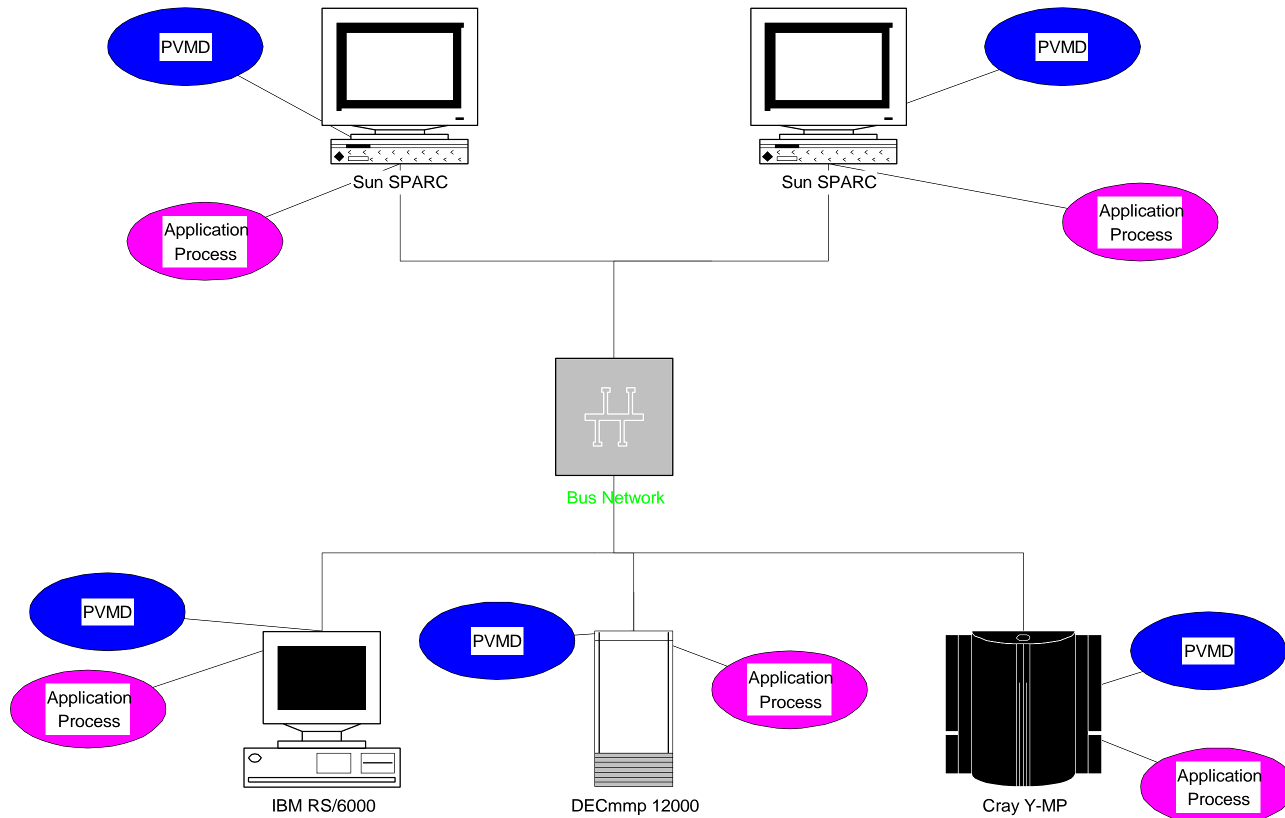
copyright 1999 Jeffrey K. Hollingsworth

PVM

- Provide a simple, free, portable parallel environment
- Run on everything
  - Parallel Hardware: SMP, MPPs, Vector Machines
  - Network of Workstations: ATM, Ethernet,
    - UNIX machines and PCs running Win*
  - Works on a heterogenous collection of machines
    - handles type conversion as needed
- Provides two things
  - message passing library
    - point-to-point messages
    - synchronization: barriers, reductions
  - OS support
    - process creation (pvm_spawn)

copyright 1999 Jeffrey K. Hollingsworth

PVM Environment (UNIX)



- One PVMD per machine
  - all processes communicate through pvmd (by default)
- Any number of application processes per node

copyright 1999 Jeffrey K. Hollingsworth

PVM Message Passing

- All messages have tags
  - an integer to identify the message
  - defined by the user
- Messages are constructed, then sent
  - pvm_pk{int,char,float}(*var, count, stride)
  - pvm_unpk{int,char,float} to unpack
- All proccess are named based on task ids (tids)
  - local/remote processes are the same
- Primary message passing functions
  - pvm_send(tid, tag)
  - pvm_recv(tid, tag)

copyright 1999 Jeffrey K. Hollingsworth

PVM Process Control

- Creating a process
  - pvm_spawn(task, argv, flag, where, ntask, tids)
  - flag and where provide control of where tasks are started
  - ntask controls how many copies are started
  - program must be installed on target machine
- Ending a task
  - pvm_exit
  - does not exit the process, just the PVM machine
- Info functions
  - pvm_mytid() - get the process task id

copyright 1999 Jeffrey K. Hollingsworth