Parallel Computing

- Basics of Parallel Computers
- Shared Memory
- SMP / NUMA Architectures
- Message Passing
- Clusters

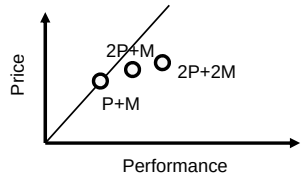
1

Why Parallel Computing

- No matter how effective ILP/Moore's Law, more is better
  - Most systems run multiple applications simultaneously
    - Overlapping downloads with other work
    - Web browser (overlaps image retrieval with display)
  - Total cost of ownership favors fewer systems with multiple processors rather than more systems w/fewer processors

Peak performance increases linearly with more processors

Adding processor/memory much cheaper than a second complete system



2

What about Sequential Code?

- Sequential programs get no benefit from multiple processors, they must be parallelized.
  - Key property is how much communication per unit of computation. The less communication per unit computation the better the scaling properties of the algorithm.
  - Sometimes, a multi-threaded design is good on uni & multi-processors e.g., throughput for a web server (that uses system multi-threading)
- Speedup is limited by Amdahl's Law
  - Speedup <= 1/(seq + (1 – seq)/proc)

    as proc -> infinity, Speedup is limited to 1/seq
- Many applications can be (re)designed/coded/compiled to generate cooperating, parallel instruction streams – specifically to enable improved responsiveness/throughput with multiple processors.

3

Performance of parallel algorithms is NOT limited by which factor

1. The need to synchronize work done on different processors.
2. The portion of code that remains sequential.
3. The need to redesign algorithms to be more parallel.
4. The increased cache area due to multiple processors.

4

Parallel Programming Models

- Parallel programming involves:
  - Decomposing an algorithm into parts
  - Distributing the parts as tasks which are worked on by multiple processors simultaneously
  - Coordinating work and communications of those processors
    - Synchronization
- Parallel programming considerations:
  - Type of parallel architecture being used
  - Type of processor communications used
- No automated compiler/language exists to automate this "parallelization" process.
- Two Programming Models exist…..
  - Shared Memory
  - Message Passing

5

Process Coordination
Shared Memory *v.* Message Passing

- Shared memory
  - Efficient, familiar
  - Not always available
  - Potentially insecure

```
global int x
process foo          process bar
begin                begin
  :                    :
 x := ...             y := x
  :                    :
end foo              end bar
```

- Message passing

  Extensible to communication in distributed systems

  Canonical syntax:

```
send(process : process_id, message : string)
receive(process : process_id, var message : string)
```

6

Shared Memory Programming Model

- Programs/threads communicate/cooperate via loads/stores to memory locations they share.
- Communication is therefore at memory access speed (very fast), and is implicit.
- Cooperating pieces must all execute on the same system (computer).
- OS services and/or libraries used for creating tasks (processes/threads) and coordination (semaphores/barriers/locks.)

7

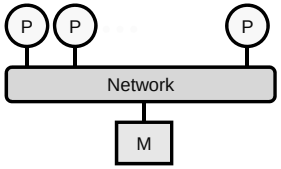
Shared Memory Code

```
fork N processes
each process has a number, p, and computes
   istart[p], iend[p], jstart[p], jend[p]
for(s=0;s<STEPS;s++) {
  k = s&1 ; m = k^1 ;
  forall(i=istart[p];i<=iend[p];i++) {
    forall(j=jstart[p];j<=jend[p];j++) {
      a[k][i][j] =  c1*a[m][i][j]   + c2*a[m][i-1][j] +
                    c3*a[m][i+1][j] + c4*a[m][i][j-1] +
                    c5*a[m][i][j+1] ; // implicit comm
    }
  }
   barrier() ;
}
```

8

Symmetric Multiprocessors

- Several processors share one address space
  - conceptually a shared memory
- Communication is *implicit*
  - read and write accesses to shared memory locations
- Synchronization
  - via shared memory locations
    - spin waiting for non-zero
  - Atomic instructions (Test&set, compare&swap, load linked/store conditional)
  - barriers



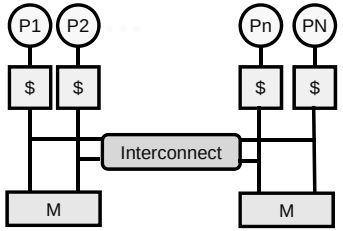
Conceptual Model

9

Non-Uniform Memory Access - NUMA

CPU/Memory busses cannot support more than ~4-8 CPUs before bus bandwidth is exceeded (the SMP "sweet spot").

To provide shared-memory MPs beyond these limits requires some memory to be "closer to" some processors than to others.



The "Interconnect" usually includes

- a cache-directory to reduce snoop traffic
- Remote Cache to reduce access latency (think of it as an L3)

- Cache-Coherent NUMA Systems (CC-NUMA):
  - SGI Origin, Stanford Dash, Sequent NUMA-Q, HP Superdome
- Non Cache-Coherent NUMA (NCC-NUMA)
  - Cray T32E

10

Message Passing Programming Model

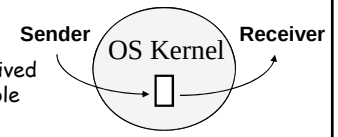
- "Shared" data is communicated using "send"/"receive" services (across an external network).
- Unlike Shared Model, shared data must be formatted into message chunks for distribution (shared model works no matter how the data is intermixed).
- Coordination is via sending/receiving messages.
- Program components can be run on the same or different systems, so can use 1,000s of processors.
- "Standard" libraries exist to encapsulate messages:
  - Parasoft's Express (commercial)
  - PVM (standing for Parallel Virtual Machine, non-commercial)
  - MPI (Message Passing Interface, also non-commercial).

11

Message Passing Issues
Synchronization semantics

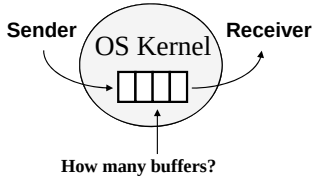
- When does a *send* /*receive* operation terminate?

Blocking (aka Synchronous):
Sender waits until its message is received
Receiver waits if no message is available



Non-blocking (aka Asynchronous):
Send operation "immediately" returns
Receive operation returns if no message is available (polling)

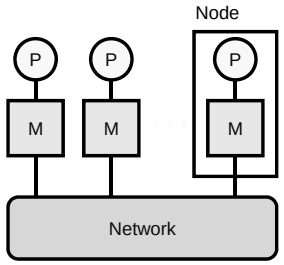
Partially blocking/non-blocking:
send()/receive() with *timeout*



12

Clustered Computers designed for Message Passing

- A collection of computers (nodes) connected by a network
  - computers augmented with fast network interface
    - send, receive, barrier
    - user-level, memory mapped
  - otherwise indistinguishable from conventional PC or workstation
- One approach is to network workstations with a very fast network
  - Often called 'cluster computers'
  - Berkeley NOW
  - IBM SP2 (remember Deep Blue?)



13

Which is easier to program?

1. Shared memory
2. Message passing

14