

Systems Architecture II

Topics

_Multiprocessors: Uniform Memory Access *
Multiprocessors: Non-Uniform Memory Access*

*This lecture was derived from material in the text (Chapter 9).

All figures from Computer Organization and Design: The Hardware/Software Approach, Second Edition, by David Patterson and John Hennessy, are copyrighted material (COPYRIGHT 1998 MORGAN KAUFMANN PUBLISHERS, INC. ALL RIGHTS RESERVED).

Notes Courtesy of Jeremy R. Johnson

Systems Architecture II

Topic 1: Multiprocessors: Uniform Memory Access

Introduction

- **Objective: To use multiple processors to improve performance. To understand the performance gain that is possible and the basic design decisions.**
 - What are the limits to the speedup that can be obtained?
 - How do parallel processors share data?
 - How do parallel processors coordinate?
 - How many processors?
- **Topics**
 - Speedup and Amdahl's Law
 - Different types of multiprocessors
 - Communication model
 - Physical connection
 - Multiprocessors connected by a single bus

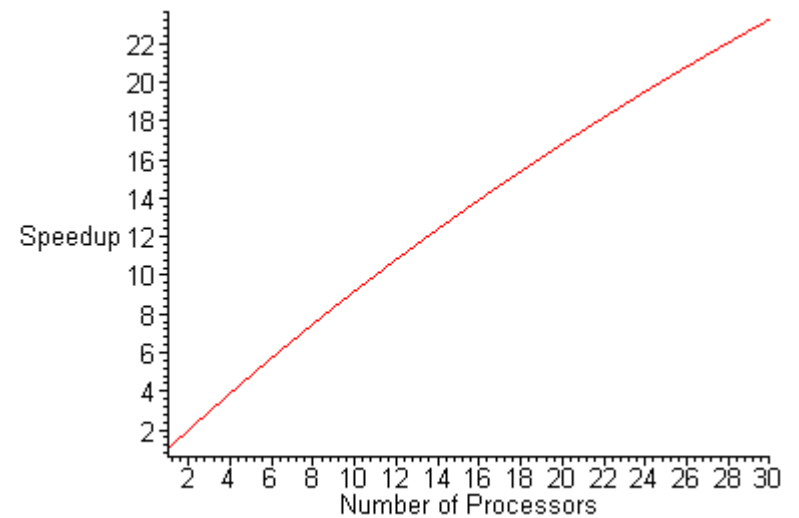
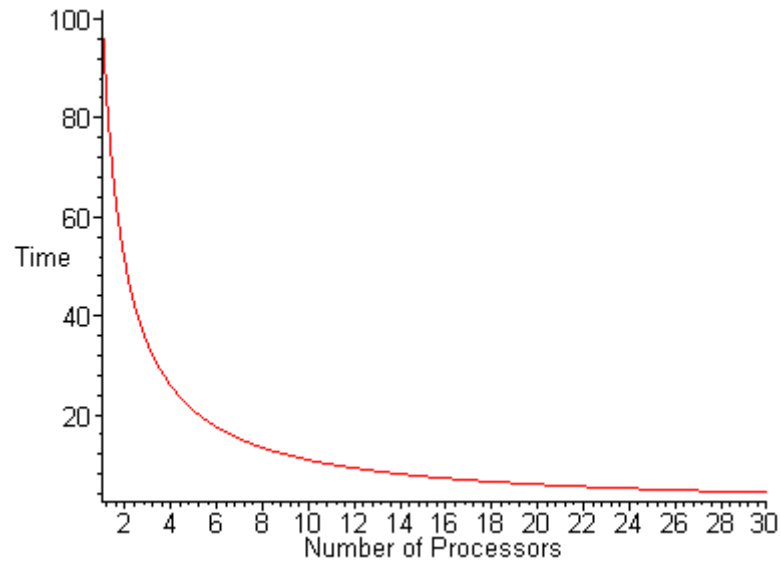
Speedup and Amdahl's Law

- **Speedup =**
Sequential execution time/Parallel execution time
- **Speedup_N =**
Sequential execution time / Execution time on N Processors
- **Amdahl's Law:**
Execution Time After Improvement =
Execution Time Unaffected +
(Execution Time Affected / Amount of Improvement)

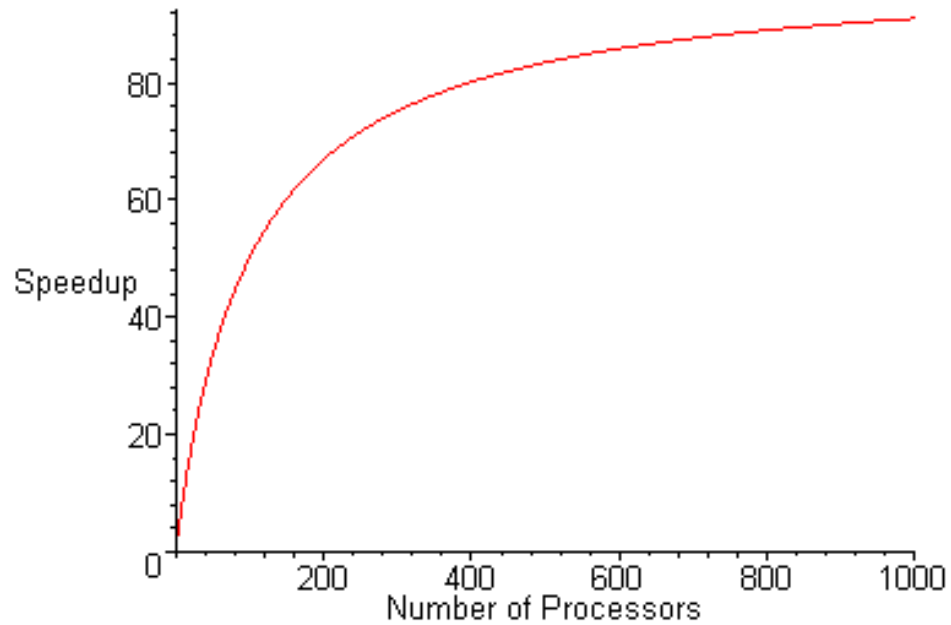
Speedup

- **Assume that the sequential execution time of a program is 100sec and that 99% of the program can benefit from parallelism**
- **The parallel execution time on N processors is:**
 - $99/N + 1$ sec
- **Speedup = $100/(99/N + 1)$**
 - Upper bound on speedup due to parallelism is 100
 - Ideally we would hope for linear speedup. In this example after a certain point additional processors will produce diminishing returns.
 - How many processors are required to obtain a speedup of 90?

Speedup vs. Number of Processors



Limiting Speedup



Multiprocessor Design Choices

- **Single Address Space (Shared Memory)**
 - Processors communicate through shared variables
 - Synchronization is used to protect processors from changing shared data simultaneously. Performed using a lock
 - Only one processor can obtain the lock at a time. Other processors must wait until the lock is released before accessing shared data.
 - After obtaining the lock the processor can safely modify shared data.
 - **Uniform Memory Access (UMA). Also called Symmetric Multiprocessor (SMP)**
 - Implementing UMA limits the number of processors
 - UMA commonly implemented using a shared bus
 - **Non-uniform Memory Access (NUMA)**
 - some memory accesses are faster than others depending on which processor asks for which word
 - Scales to larger number of processors
 - More difficult to effectively program

Multiprocessor Design Choices

- **Distributed Memory (separate private memories)**
 - **Processors communicate through message passing**
 - send and receive primitives
 - **Synchronization performed using send and receive**
 - **Processors connected via a network**
 - In the extreme a cluster of workstations can communicate over a local area network

Category	Choice		Number of Processors
Communication Model	Shared Address	NUMA	8-256
		UMA	2-64
	Message Passing		8-256
Physical Connection	Network		8-256

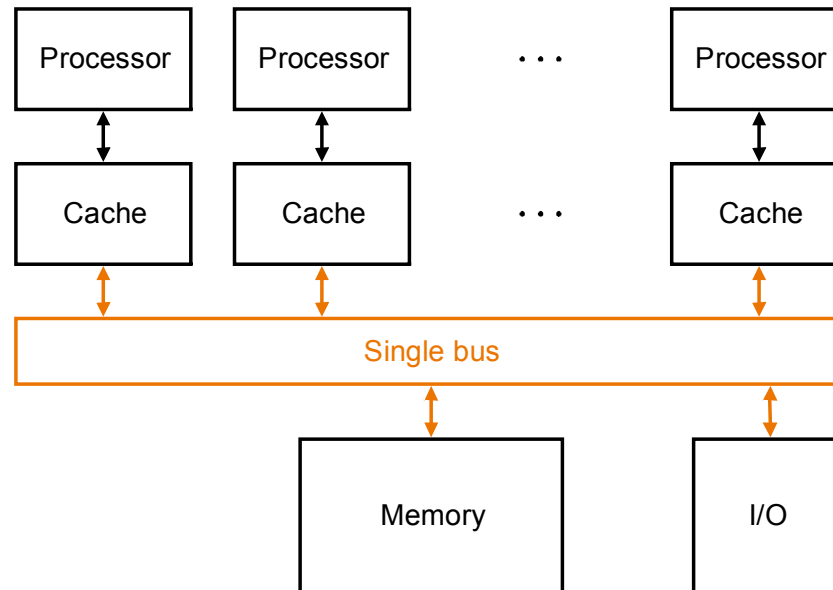
Shared Memory Programming

- **Example: Sum 100,000 numbers (Assume 10 processors)**
- **Each processor executes the same program (Pn = processor number), Array A contains shared data.**

```
sum[Pn] = 0;
for (i=10000*Pn; i < 10000*(Pn + 1); i++)
    sum[Pn] = sum[Pn] + A[i];

half = 10;
repeat
    synch(); /* wait for partial sum completion */
    if (half % 2 != 0 && Pn == 0)
        sum[0] = sum[0] + sum[half - 1];
    half = half/2;
    if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn + half];
until (half == 1); /* exit with final sum in sum[0] */
```

Multiprocessors Connected by a Single Bus



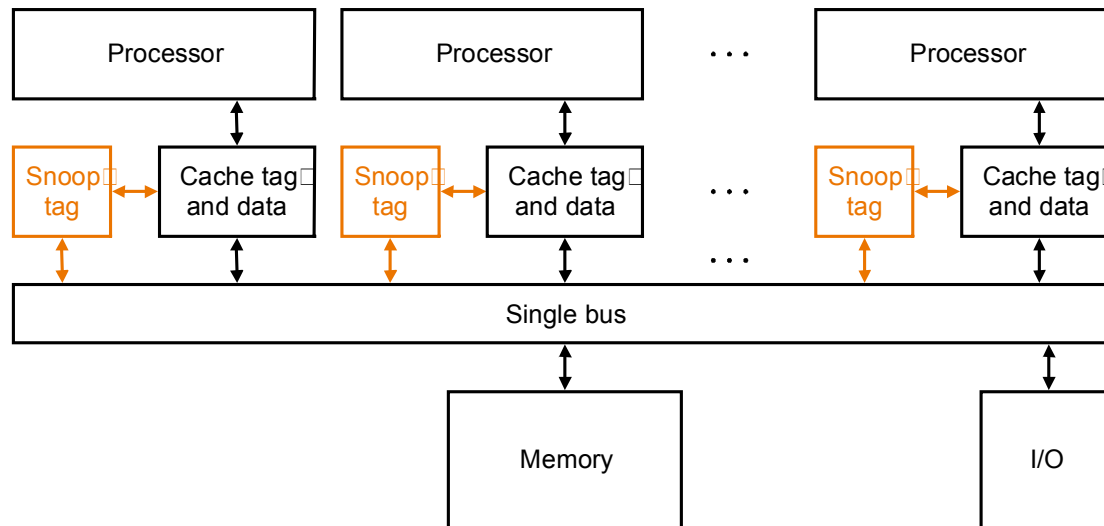
Cache Coherency

- **Need to make sure that processor caches are consistent with each other and memory**
 - exclusive access for a write
 - processors must have most recent copy when reading

Time	Event	Cache A	Cache B	X (Memory)
0				1
1	A reads X	1		1
2	B reads X	1	1	1
3	A stores 0 in X	0	1	0

- **Snooping: All cache controllers monitor, or snoop, on the bus to determine whether or not they have a copy of a shared block**
 - When a write occurs either invalidate or update local copy. Write-invalidate reduces demands on bus bandwidth; however, write-update makes new values appear in cache sooner, which can reduce latency.

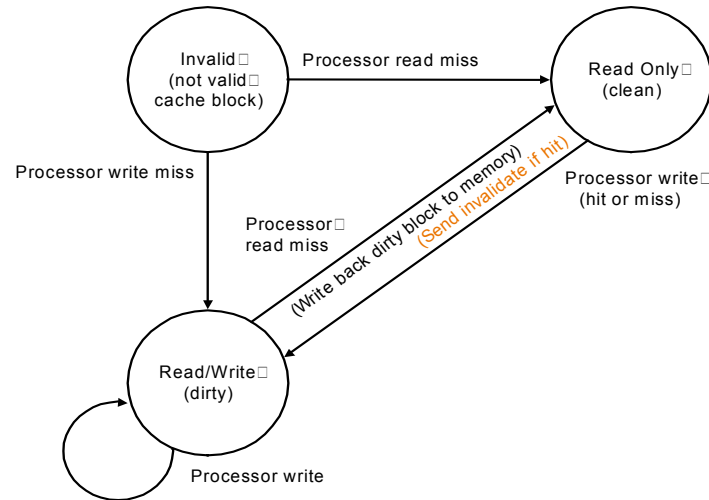
Single-bus Multiprocessor using Snooping Cache Coherency



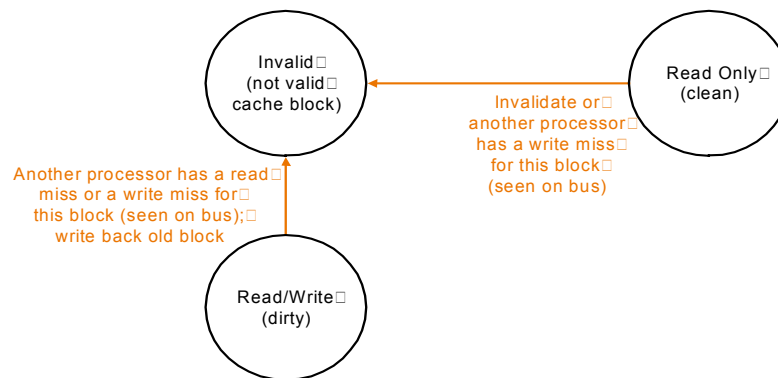
Implementation of Invalidate Protocol

- Assume Write-Back Cache
- Each cache block in 1 of 3 States:

- Read Only
- Read/Write
- Invalid



a. Cache state transitions using signals from the processor

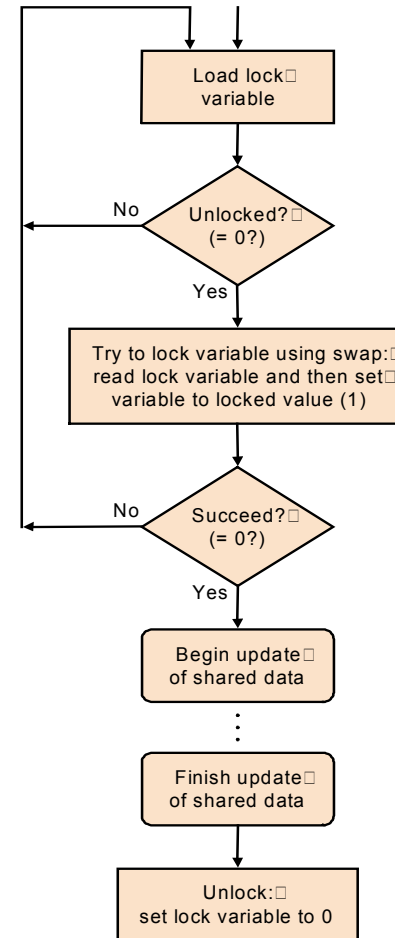


b. Cache state transitions using signals from the bus

Synchronization using Coherency

- Atomic swap operation
- Spin on local copy of lock
- unlocked = 0
- locked = 1

Step	P0	P1	P2	Bus
1	has lock	spin	spin	None
2	sets lock to 0	spin	spin	Invalidate
3		cache miss	cache miss	service P2
		wait while bus busy	Lock = 0	cache miss for P2 satisfied
		Lock = 0	Swap	cache miss for P1 satisfied
		swap	value = 0 send 1	Invalidate from P2
		value = 1, send 1	owns lock	Invalidate from P1
		spin		None



Systems Architecture II

Topic 2: Multiprocessors: Non-Uniform Memory Access

Introduction

- **Objective:** To introduce multiprocessors where the cost of memory access varies depending on the processor and memory address accessed (NUMA). Introduce distributed memory and message passing. To compare the cost/performance tradeoffs of UMA vs. NUMA.
- **Topics**
 - **Programming and Communication Models**
 - Shared memory vs. message passing
 - **Different types of multiprocessors**
 - Communication model
 - Physical connection
 - **Multiprocessors connected by a network**
 - cache coherency
 - network topology
 - **Cost/performance tradeoffs (UMA vs. NUMA)**

Multiprocessor Design Choices

- **Single Address Space (Shared Memory)**
 - Processors communicate through shared variables
 - Synchronization is used to protect processors from changing shared data simultaneously. Performed using a lock
 - Only one processor can obtain the lock at a time. Other processors must wait until the lock is released before accessing shared data.
 - After obtaining the lock the processor can safely modify shared data.
 - **Uniform Memory Access (UMA). Also called Symmetric Multiprocessor (SMP)**
 - Implementing UMA limits the number of processors
 - UMA commonly implemented using a shared bus
 - **Non-uniform Memory Access (NUMA)**
 - some memory accesses are faster than others depending on which processor asks for which word
 - Scales to larger number of processors
 - More difficult to effectively program

Multiprocessor Design Choices

- **Distributed Memory (separate private memories)**
 - **Processors communicate through message passing**
 - send and receive primitives
 - **Synchronization performed using send and receive**
 - **Processors connected via a network**
 - In the extreme a cluster of workstations can communicate over a local area network

Category	Choice		Number of Processors
Communication Model	Shared Address	NUMA	8-256
		UMA	2-64
	Message Passing		8-256
Physical Connection	Network		8-256

Why network multiprocessors?

- **Single bus designs are attractive because they are simple.**
- **But they are limited because the three desirable bus characteristics are incompatible: high bandwidth, low latency and long length.**
- **A single memory module (which is shared) has also limited bandwidth.**
- **Shared memory vs. distributed memory.**
 - **A single address space implies implicit communication with loads and stores.**
 - **Distributed Memory (separate private memories) implies explicit communication with send and receive primitives.**
 - **Synchronization performed using send and receive.**
 - **Processors connected via networks with various topologies.**
- **Shared virtual memory**
 - **A software layer on top of distributed memory that provides an illusion of shared memory. This is very costly in practice.**

Distributed Memory Programming

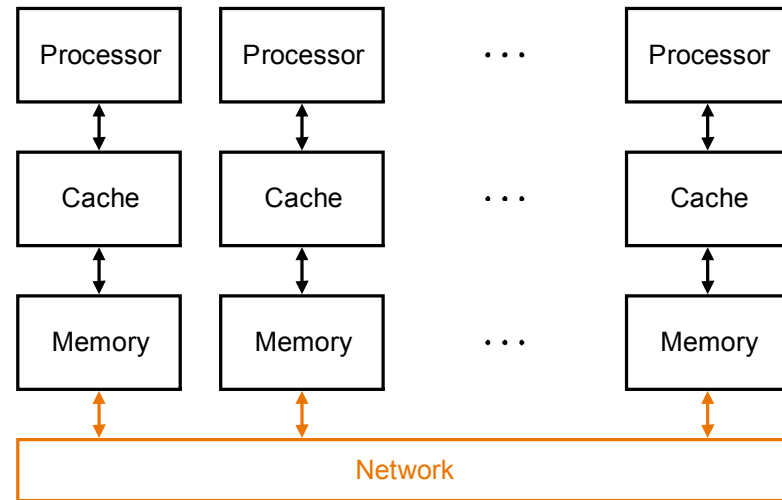
- **Example: Sum 100,000 numbers (Assume 100 processors)**
- **Partition global array, A, into 100 equal parts and distribute each part to a different processor:**

```
sum = 0;
for (i=0; i < 1000; i++)
    sum = sum + A[i];
```

```
half = 100; limit = 100;
repeat
    half = (half+1)/2; /* send vs. receive dividing line */
    if (Pn >= half && Pn < limit) send(Pn - half, sum);
    if (Pn <= limit/2 - 1) sum = sum + receive();
    limit = half; /* upper limit of senders */
until (half == 1); /* exit with final sum */
```

Multiprocessors Connected by a Network

- **Single address space vs. private memories**
 - send/receive easier and cost is explicit
 - load/store less overhead
 - Memory allocation
- **Allow shared data to appear on the processor where it is stored and the processor requesting it**
 - Cache coherency
 - Memory migration
- **Different Network topologies**



Network Cache Coherency

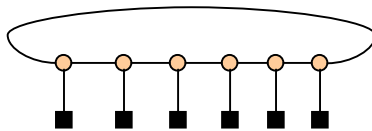
- **Can not use bus-snooping**
- **Directories: Logically there is a single directory which keeps the state of every block in main memory**
 - Information: which caches have a copy of the block, whether the block is dirty, etc.
 - Directory can be distributed so that different requests go to different memories. This reduces contention and allows scalability.
 - Need mechanism to detect when there is a write to a shared block
 - Must inform processors when a local cache must be invalidated or updated.
 - Directory controller sends explicit commands to each processors that has a copy of the data.
- **Coherency at the cache or memory level is possible**
 - Second level of coherence to main memory of each processor: allow blocks of main memory to migrate – hardware memory allocation.

Network Topologies

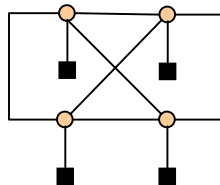
- **The straightforward way to connect processor-memory nodes is with a link between every node. However this is expensive.**
- **Range of alternatives between a single bus and a link between every node.**
- **All networks consist of switches that can be connected to processor-memory nodes and other switches.**
- **Performance measure:**
 - **total network bandwidth (best case): bandwidth of a link \times number of links**
 - **bisection bandwidth: (closer to worst case): divide machine into two parts and multiply the bandwidth of a link \times the number a links between the two parts (choose split to minimize performance)**

Example Topologies

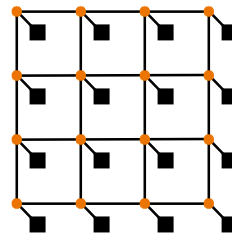
- **Ring: TB = p, BB = 2**
- **Fully connected network: TB = p(p-1)/2, BB = (p/2)²**
- **2D Grid or Mesh: TB = 2p, BB = \sqrt{p}**
- **n-cube: 2^n nodes, $n2^{n-1}$ edges. TB = $p \lg(p)$, BB = $p/2$**



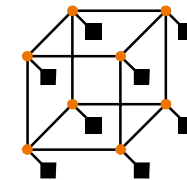
Ring



Fully Connected



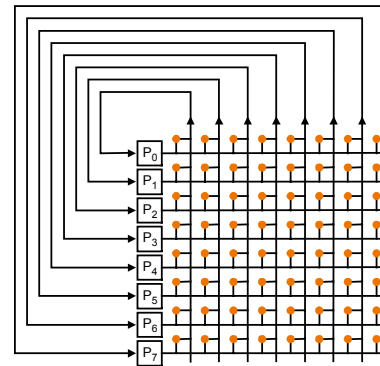
a. 2D grid or mesh of 16 nodes



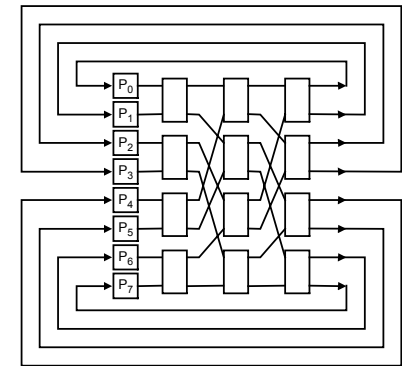
b. n-cube tree of 8 nodes ($8 = 2^3$ so $n = 3$)

Multistage Networks

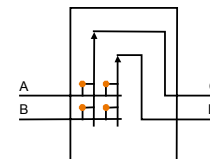
- Not all nodes in the network contain processors (some only contain switches): Decrease distance and improve performance
- Called multistage networks since multiple steps may be required to route a message.
- Examples
 - crossbar (p^2 switches)
 - omega ($2plg(p)$ switches)



a. Crossbar



b. Omega network



c. Omega network switch box

Cost Performance of UMA vs. NUMA

- **Network has smaller initial cost**
- **Network cost scales somewhat more quickly than bus**
- **Performance of both machines scales linearly until bus reaches limit**
- **NUMA has consistent cost per node**
- **Bus based architecture has a plateau of optimal cost per node (better than NUMA)**

