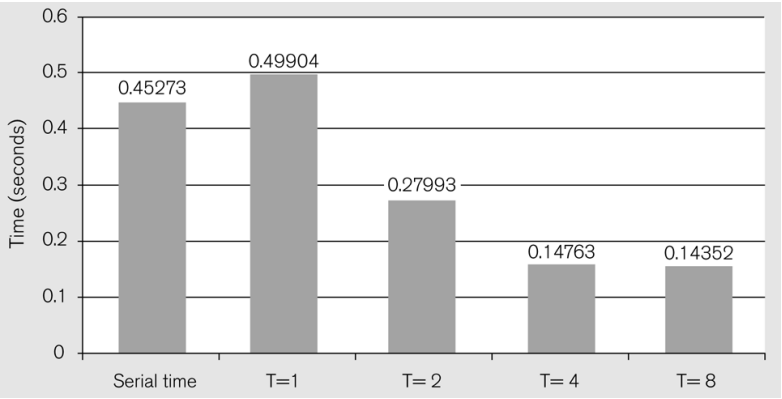
1. Fourth Try and Final Solution:

```
 1  struct padded_int
 2  {
 3    int value;
 4    char padding[60];
 5  } private_count[MaxThreads];
 6
 7  void count3s_thread(int id)
 8  {
 9    /* Compute portion of the array this thread should
         work on */
10    int length_per_thread=length/t;
11    int start=id*length_per_thread;
12
13    for(i=start; i<start+length_per_thread; i++)
14    {
15      if(array[i] == 3)
16      {
17        private_count[id]++;
18      }
19    }
20    mutex_lock(m);
21    count+=private_count[id].value;
22    mutex_unlock(m);
23  }
```

Performance of the Fourth Try code:



a) Why did the change help?

2. Goal of textbook "help you write good parallel programs, by which we mean parallel programs with the four characteristics:

- They are correct
- They achieve good performance
- They are scalable to large number of processors
- They are portable across a wide variety of parallel platforms" (*performance portability*)

a) Why might it be important even on a desktop machine to scale to a large number of processors?

b) What about the "Fourth Try Code" is not scalable to a large number of processors?

c) Why is performance portability (i.e., runs well on different parallel machines) difficult?

Chapter 2 goals:

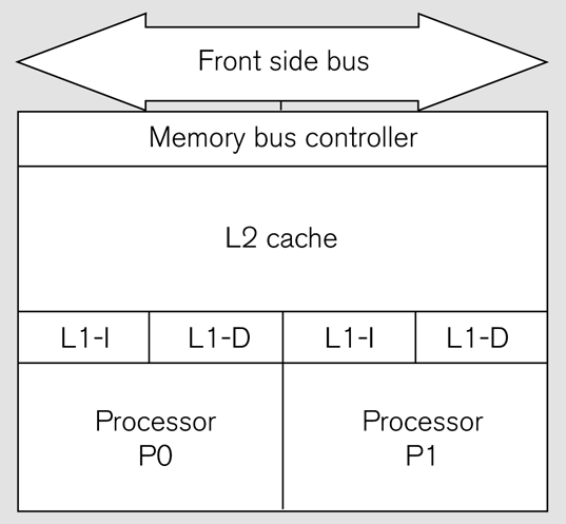
- illustrate the diversity of parallel computers,
- consider ways to abstract away from details to support performance portability (PRAM and CTA abstract models)
- discuss three major communication mechanisms presented to programmers (shared memory, one-sided communication, and message passing)

Six "current" parallel computers to show diversity and topics relevant to programmers.

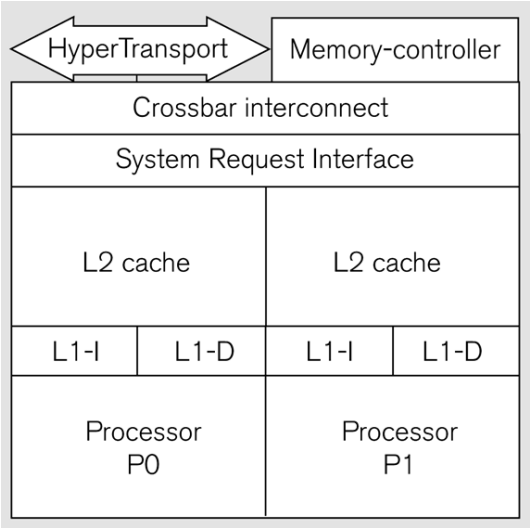
Chip Multiprocessors:

Programmers point of view: both designs implement a coherent shared memory

Intel Core Duo



AMD64 Dual Core Opteron

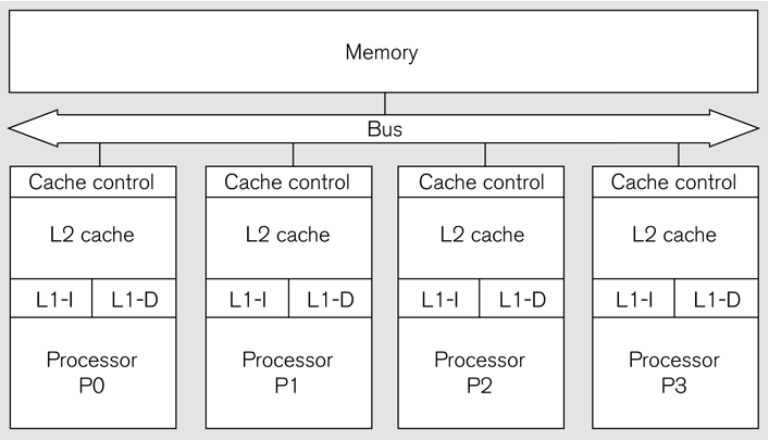


a) Which design allows a processor-core more "private" memory (i.e., dedicate cache)?

b) Which design allows a processor running a process with high memory utilization to use more of the cache?

c) Intel Core Duo uses a MESI (Modified, Exclusive, Shared, Invalid) cache coherency protocol. To write a shared cache line, the processor must first obtain Exclusive use of the line. This optimized between processors on a single chip, but can be slow if multiple processor chips on the system. AMD64 Dual Core uses a MOESI cache coherency protocol with an additional "Owned" state that allows cache values to be shared among processors over the System Request Interface (SRI) even when the RAM copy is *stale* (outdated). Why might this be faster on multiple processor chips are on the system?

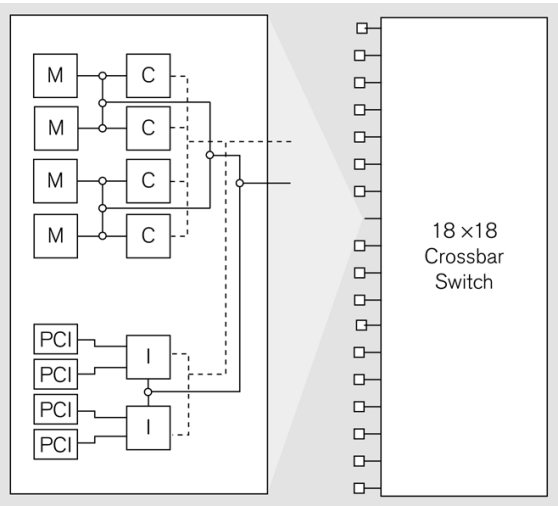
Symmetric Multiprocessor Architecture: (SMP) all processors access a single logical memory. Typically, each processor sees a consistent view of memory by use of snoopy caches on the bus (or SRI of AMD) with some write-invalidate cache coherency protocol (e.g., MESI or MOESI).



a) What is the bottleneck in the SMP architecture that limits the system to 10s of processors?

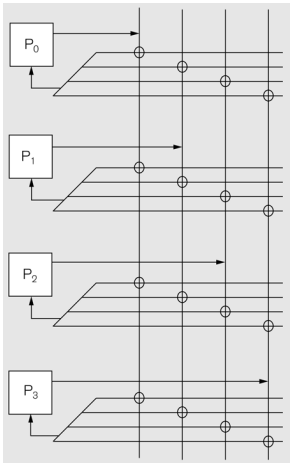
b) Why are large L2 caches helpful?

Sun Fire E25k (extreme SMP example):



- Up to 72 processor
- 150-MHz Sun Fireplane of 3 18x18 crossbars for address, response and data, and 18 snoopy buses
- Access latency to shared memory equal for all processors
- Shared memory of 1.15 TB
- 18 E25K boards each containing:
  - 4 Ultra SPARC IV Cu processors
  - 16 GB of "local" memory
  - 18 snoopy buses using a directory-base cache coherency protocol (dashed lines)

Example of 4x4 crossbar switch



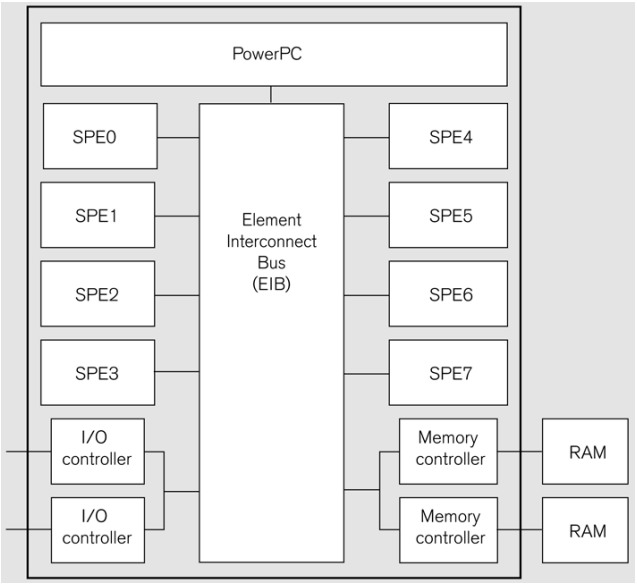
c) Why would a processor accessing "local" memory be faster than access to a "nonlocal" memory?

d) Why is an 18 x 18 crossbar switch about the maximum size?

Heterogeneous Chip Designs: Augment a standard processor with one or more specialized compute engines (*attached processors*). General idea:

- standard processor performs the hard-to-parallelize portion of the computation that is probably fast enough
- attached processors perform the compute-intensive portion of the computation

Examples: Graphics processing units (GPUs), Field programmable gate arrays (FPGAs), and game cell processor



Sony Playstation Cell processor:

- 64-bit PowerPC core (the standard processor)
- 8 specialized cores SPEs (synergistic processing elements) executing vector instructions
- high-speed EIB bus connecting the SPEs

Difficult to program since designed for performance.

vector instruction - one operation performed on several data values in parallel. SPEs have 128-bit wide data paths support:

- 8-bit integer operations on 16 values,
- 16-bit integer operations on 8 values, or
- 32-bit integer or fl. pt. operations on 4 values.

Clusters: Parallel computers made from commodity parts:

- nodes - boards containing processor(s), RAM memory, and often disks
- interconnection of nodes using commodity networking form (Gigabit ethernet, Myrinet, Infiniband, etc.)

Key characteristics:

- tremendous price/performance advantage over "supercomputer"
- memory is not shared among processor nodes - communication done by message passing

Blade server - pre-packaged cluster from a manufacture.

- blade - board containing a few processor chips, RAM, disk, and communication ports
- blades fit into rack to form blade server.

Supercomputers: - scaled up cluster with fancy interconnection networks and specially designed hardware.