# Symmetric Multiprocessors

Symmetric Multiprocessors are said to be tightly coupled

- Also called shared memory multiprocessor
- A type of MIMD system

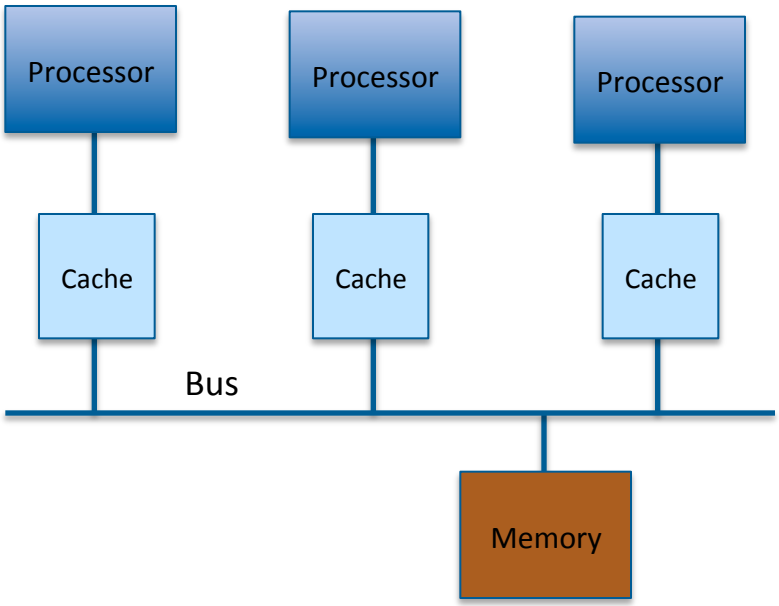
An SMP architecture treats all processors equally

- *Symmetric* implies the processors are logically interchangeable
- The OS divides and distributes the work to processors

Programs can be written to run faster on SMPs

Programs optimized for SMP will suffer if run on uniprocessor

# Symmetric Multiprocessors

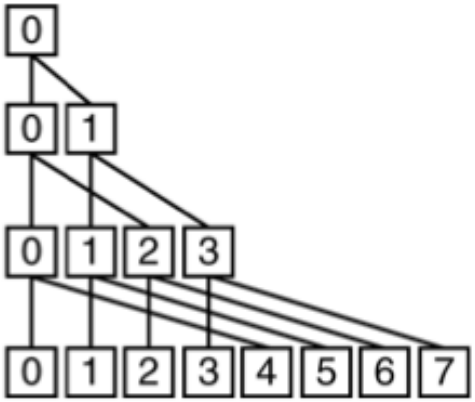



SMPs use a common shared bus

- The bus may become a bottleneck
- Processors must make good use of their internal caches
- Cache misses cause stalls and contention for the bus

# Example: Sum Reduction on SMP

- To illustrate the idea, assume an 8-element array
- Assume there are 4 processors
- Each will add 2 elements




```
Sum[P0] = A[0] + A[1]
Sum[P1] = A[2] + A[3]
Sum[P2] = A[4] + A[5]
Sum[P3] = A[6] + A[7]

Then use half the processors (2)
Sum[P0] = Sum[P0] + Sum[P2]
Sum[P1] = Sum[P1] + Sum[P3]

Finally use one processor: Sum[P0] = Sum[P0] + Sum[P1]
```

# Example: Sum Reduction on SMP

- Sum 100,000 numbers on 100 processor UMA
  - Each processor has ID: 0 ≤ Pn ≤ 99
  - Partition 1000 numbers per processor
  - Initial summation on each processor

```
sum[Pn] = 0;
  for (i = 1000*Pn;
      i < 1000*(Pn+1); i = i + 1)
    sum[Pn] = sum[Pn] + A[i];
```

```
Sum[0] = A[0] + … + A[999]
.
.
.
Sum[99] = A[99000]+ … + A[99999]
```

# Example: Sum Reduction on SMP

- Now need to add these 50 partial sums
  - Reduction: divide and conquer
  - Half the processors add pairs, then quarter, …
  - 50, then 25, then 12, then 6, then 2, then 1
  - Using ½ may yield an odd number
    - P0 takes care of the left over value
  - Need to synchronize between reduction steps

# Example: Sum Reduction on SMP

```
half = 100;
do
  synch();
  if (half%2 != 0 && Pn == 0)
    sum[0] = sum[0] + sum[half-1];
    /* Conditional sum needed when half is odd;
       Processor0 gets missing element */
  half = half/2; /* dividing line on who sums */
  if (Pn < half) sum[Pn] = sum[Pn] + sum[Pn+half];
while (half > 1);  // exit with final sum in Sum[0]
```

synch() insures that all required partial sums have been produced

Private variables, such as *half* or a loop index, are local to each processor