More jargon...

- *Deadlock* is a situation wherein two or more competing actions are waiting for the other to finish, and thus neither ever does.

  Example: A thread may require exclusive access to a table, and in order to gain exclusive access it asks for a lock. If one thread holds a lock on a table and attempts to obtain the lock on a second table that is already held by another thread, this may lead to deadlock if the second thread then attempts to obtain the lock that is held by the first thread.

- *Livelock* is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing.

...jargon

- *Race condition* is a flaw in a program whereby the output of the program is unexpectedly and critically dependent on the sequence or timing of other events.

- *Efficiency* is defined by

$$E = s/N,$$

  where s is the observed speedup, and N is the number of processors.

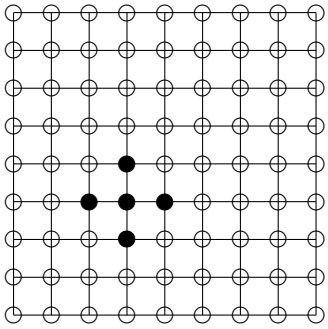
  Ideally $E = 1 = 100\%$.

5 Cost of communication

5.1 Local communication

Cost of communication: 2D finite differences

```
do j=1,n
  do i=1,n
    unew(i,j) = b*u(i,j) + &
     c*(u(i+1,j)+u(i-1,j)+u(i,j+1)+u(i,j-1))
  end do
end do
```



Simplified communications cost model

N = number of processors
t_c = time spent by one fp operation
t_s = communication startup time (latency)
t_w = time spent by communicating one fp number

Work done by processor in one iteration (or time step):

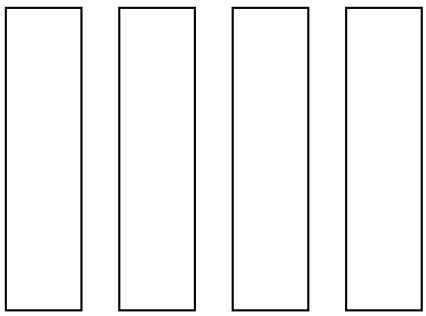
$$T = T_{comp} + T_{comm},$$

where

$T_{comp} = t_c \cdot$ number of arithmetic operations
$T_{comm} = t_s + t_w \cdot$ number of fp numbers transferred

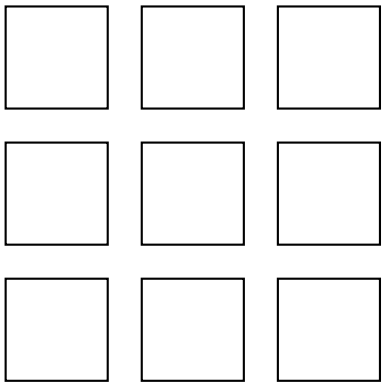
---

1D domain decomposition



Each task needs $2n$ values from the two neighboring tasks. (Assume periodic boundary conditions).

2D domain decomposition



Each task needs $4\frac{n}{\sqrt{N}}$ values from the four neighboring tasks.

Total cost (execution time)

1D decomposition:

$$T = T_{comp} + T_{comm} = t_c \frac{n^2}{N} + 2(t_s + t_w n)$$

2D decomposition:

$$T = T_{comp} + T_{comm} = t_c \frac{n^2}{N} + 4(t_s + t_w \frac{n}{\sqrt{N}})$$

Isoefficiency analysis

Assume that the number of processors N is increaseing. How should the problem size increased to maintain constant efficiency?

$$E = \frac{T_1}{N T_N} = \text{constant},$$

where

$$T_k = \text{execution time using } k \text{ processors.}$$

Parallel algorithm is said to be *scalable* if for constant E the problem size should increase as $\mathcal{O}(N)$.

Isoefficiency of 1D domain decomposition

$$\begin{aligned} E_{1D} &= \frac{t_c n^2}{N\left(t_c \frac{n^2}{N} + 2(t_s + t_w n)\right)} \\ &= \frac{t_c n^2}{t_c n^2 + 2 t_s N + 2 t_w N n} \end{aligned}$$

Thus $n \sim N$, and the computational work to be done is $\mathcal{O}(n^2) = \mathcal{O}(N^2)$.

Thus, the problem is *not* scalable.

Isoefficiency of 2D domain decomposition

$$\begin{aligned} E_{2D} &= \frac{t_c n^2}{N\left(t_c \frac{n^2}{N} + 4(t_s + t_w \frac{n}{\sqrt{N}})\right)} \\ &= \frac{t_c n^2}{t_c n^2 + 4 t_s N + 4 t_w n \sqrt{N}} \end{aligned}$$

Thus $n \sim \sqrt{N}$, and therefore the problem is scalable as the computational work to be done is $\mathcal{O}(n^2) = \mathcal{O}(N)$.

5.2 Global communication

Global communication

- A global communication operation is one in which many tasks must participate.
- When such operations are implemented, it may not be sufficient simply to identify individual producer/consumer pairs.
- Such an approach may result in too many communications or may restrict opportunities for concurrent execution.

Example

Parallel *reduction operation*, that is, an operation that reduces N values distributed over N tasks using a commutative associative operator (such as addition):

$$s = \sum_{i=0}^{N-1} x_i$$

Solution #1

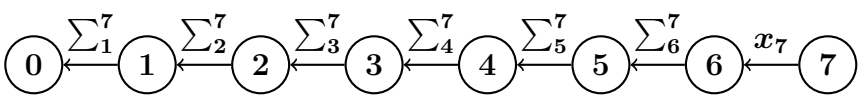
- Let us assume that a single "master" task requires the result s.
- Taking a purely local view of communication, we recognize that the manager requires values $x_0,\ x_1, ..., x_{N-1}$ from tasks $0, ..., N-1$.
- Define a communication structure that allows each task to communicate its value to the master independently. The master would then receive the values and add them into s.
- However, because the manager can receive and sum only one number at a time, this approach takes $\mathcal{O}(N)$ — not good!

Solution #2

- We can distribute the summation of the N numbers by making each task i, $0 < i < N-1$, compute the sum:

$$s_i = x_i + s_{i-1}.$$

- The communication requirements associated with this algorithm can be satisfied by connecting the N tasks in a one-dimensional array.



...solution #2

- Task $N-1$ sends its value to its neighbor in this array. Tasks 1 through $N-2$ each wait to receive a partial sum from their right-hand neighbor, add this to their local value, and send the result to their left-hand neighbor.
- Task 0 receives a partial sum and adds this to its local value to obtain the complete sum.
- This algorithm distributes the $N-1$ communications and additions, but permits concurrent execution only if multiple summation operations are to be performed.
- A single summation still takes $N-1$ steps.

The solution: Divide and conquer

- Opportunities for *concurrent computation and communication* can often be uncovered by applying a problem-solving strategy called *divide and conquer*.
- To solve a problem (such as summing N numbers), we seek to partition it into two or more simpler problems of roughly equivalent size (e.g., summing $N/2$ numbers).
- This process is applied recursively to produce a set of subproblems that cannot be subdivided further (e.g., summing two numbers).

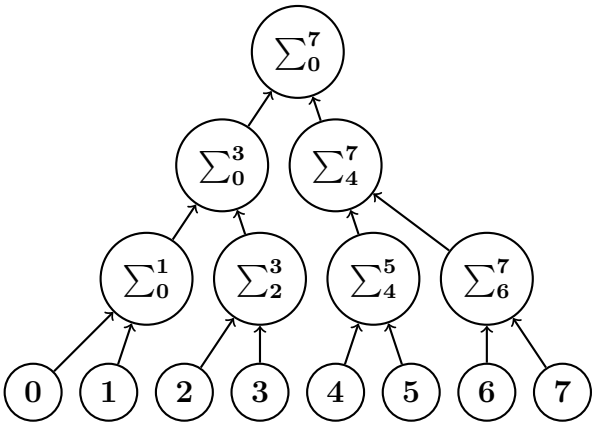
Divide and conquer

- The divide-and-conquer technique is effective in parallel computing when the subproblems generated by problem partitioning can be solved *concurrently*.
- For example, in the summation problem, we can take advantage of the following identity ($N = 2^n$):

$$\sum_{i=0}^{2^n-1} x_i = \sum_{i=0}^{2^{n-1}-1} x_i + \sum_{i=2^{n-1}}^{2^n-1} x_i.$$

- The two summations on the right hand side can be performed concurrently. They can also be further decomposed if $n > 1$, to give a tree structure. Summations at the same level in this tree of height $n = \log N$ can be performed concurrently, so the complete summation can be achieved in $\log N$ rather than N steps.

Tree structure divide-and-conquer summation



Tree structure divide-and-conquer summation

- The N (here $N = 8$) numbers located in the tasks at the bottom are communicated to the tasks in the row immediately above.
- These each perform an addition and then forward the result to the next level.
- The complete sum is available at the root of the tree after $\log N$ steps.
- The result can be broadcasted to all nodes ("allreduce" operation) again in $\log N$ steps using the same communication pattern in reverse direction.