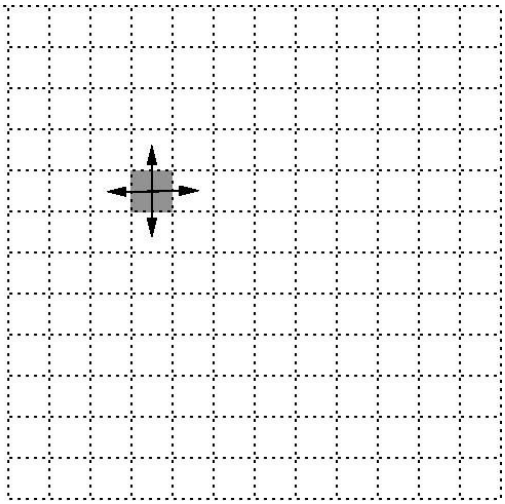
Performance Implications of Problem Decomposition

Abstract

This module demonstrates that how a problem is distributed between tasks can affect the amount of data that must be communicated between them. The specific example used is a 2D mesh of square cells with a nearest neighbor stencil communication pattern in which each cell exchanges data with its four neighbors.

Sample problem

Simulations can require splitting up a problem domain into cells, running the calculation in each cells, and then sharing information between neighboring cells. For example, a weather simulation might split a region of the Earth's surface into a 2D grid of cells, essentially dividing the region by their longitude and latitude. The simulation would then proceed in time steps, each consisting of an exchange of relevant data (surface temperature, cloud cover, humidity, etc) and then calculating the weather conditions forward for a period of time. This decomposition and the pattern of data sent from a specific node is shown in the following:



This communication is sometimes called a 4-point stencil because each cell is exchanging data with 4 neighbors. Other stencil patterns are also possible, such as an 8 point stencil which includes diagonal neighbors or stencils that include some cells that are not immediate neighbors.

First decomposition: Strips

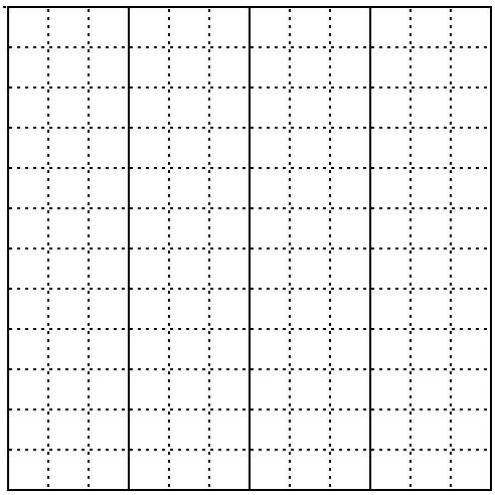
A nice aspect of this kind of simulation is that the calculations for each cell during a timestep can be done in parallel since they are independent until the next communication. Thus,

if there are as many processors as cells, each processor can be assigned its own cell, maximizing the amount of parallelism.

When there are fewer processors than grid cells, then the grid cells must be partitioned between the processors. This should be done so that each processor is assigned the same amount of computational work to achieve an even load balance. Assuming that each cell requires the same amount of computation, then each processor should be assigned the same number of cells. (This assumption is reasonable if the cells represent the same area, but it is not necessarily true if the computational work of a cell varies based on its weather conditions.)

In addition to balancing the computational work, it is also desirable to minimize the amount of communication to reduce the application's pressure on the network. As we shall see, the amount of communication depends on the decomposition used.

As a first example, suppose that each processor is assigned a vertical strip of cells comprising a number of columns that leads to an even split of the cells. This decomposition is as follows:



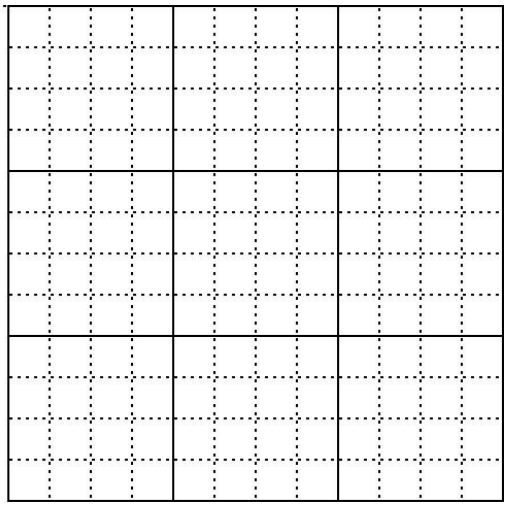
Let us assume that the grid of cells has dimensions $n \times n$, there are p processors, and that the amount of communication between each pair of neighboring cells is the same. The last assumption means that the communication between processors is proportional to the number of cells on the boundary between them. In addition, for simplicity, we assume that the numbers all divide nicely so that no columns are split between processors and each processor gets the same number of columns.

With these assumptions, each of the p processors is assigned n/p columns, containing a total of n^2/p cells. Of the columns assigned to a processor, the first and last column's data must be shared with the processors responsible for neighboring columns. All the other communication is between cells assigned to the same processor and thus does not require network traffic. Thus, the number of columns of cells whose data is communicated between processors is $2p - 2$, where the -2 comes from the very first and very last column not being communicated since they are on

the edge of the simulated region. This makes the total number of cells whose data is communicated $2pn - 2n$.

Second decomposition: Square blocks

A second possible way to assign cells to processors is to assign each processor a rectangular block of cells. Again assuming that the numbers divide nicely, the processors can be (logically) arranged into a $\sqrt{p} \times \sqrt{p}$ grid, each assigned a $n/\sqrt{p} \times n/\sqrt{p}$ subgrid of the cells as shown in the following:



As with the decomposition into strips, this assigns each processor $(n/\sqrt{p})^2 = n^2/p$ cells and the computational load is balanced.

To determine the amount of communication, it is possible to calculate the number of cells on the boundary of a processor's assignment, multiply by the number of blocks, and then subtract to compensate for blocks along the boundary, but this turns out not to be necessary. It is simpler if we separate the communication into the parts travelling in each dimension (x-dimension and y-dimension) and then count the number of subgrid boundaries in each dimension. In each of the 2 dimensions, there are $\sqrt{p} - 1$ boundaries, each involving the communication of n cells worth of data in each of 2 directions (+ and -). Thus, the total number of cells whose data is communicated is $4(\sqrt{p} - 1)n$.

Focusing on just the highest-order terms, we see that the strip decomposition requires approximately $2pn - 2n$ communication while the square block decomposition requires approximately $4\sqrt{p}n$. Thus, the block decomposition will be more communication efficient when p is sufficiently large ($\sqrt{p} > 2$, i.e. $p > 4$). While it is likely more complicated to implement, this suggests that the block decomposition will often perform better.