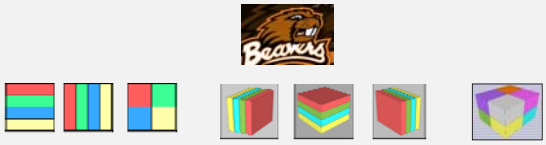
1

Parallel Program Design Patterns and Strategies

Mike Bailey

mjb@cs.oregonstate.edu

Oregon State University



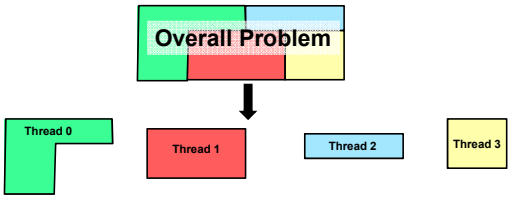
2

Contents

The goal of this section is to look at some of the common design and programming patterns one encounters in parallel programming and to understand some of the nuances one encounters.

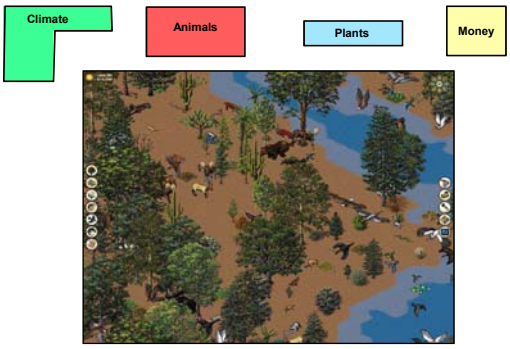
3

The Functional Decomposition Design Pattern



4

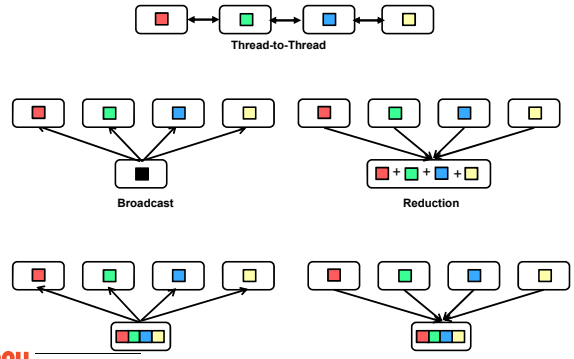
The Functional (or Task) Decomposition Design Pattern



Credit: Maxis (Sim Park)

5

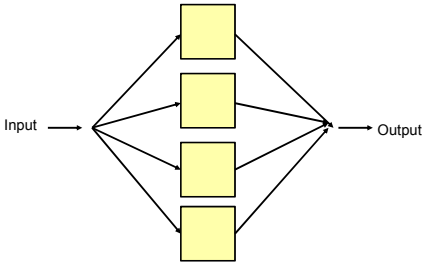
Task Distribution Design Patterns for Parallelism



6

Task Distribution Design Patterns for Parallelism

Decentralized (Peer)

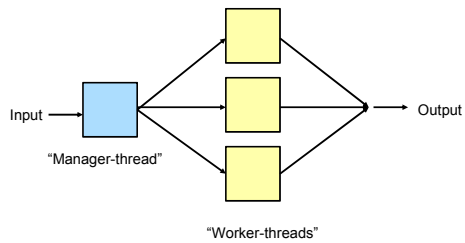


"Peer-threads"

Task Distribution Design Patterns for Parallelism

7

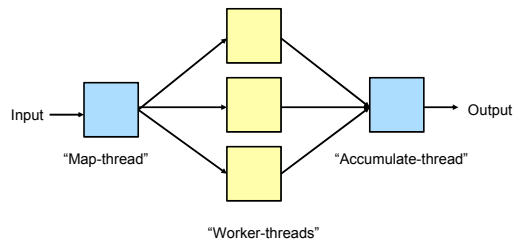
Manager / workers



Task Distribution Design Patterns for Parallelism

8

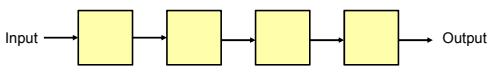
Map-Reduce



Task Distribution Design Patterns for Parallelism

9

Pipeline



Requires some sort of queue between the stages

Multicore Block Data Decomposition: 1D Heat Transfer Example

10

You have a steel bar. Each section of the bar starts out at a different temperature. There are no incoming heat sources or outgoing heat sinks (i.e., ignore boundary conditions). Ready, go! How do the temperatures change over time?

The fundamental differential equation here is: $\rho C \frac{\partial T}{\partial t} = k(\frac{\partial^2 T}{\partial x^2})$

where:
ρ is the density in kg/m^3
C is the specific heat capacity measured in Joules / (kg·°K)
k is the coefficient of thermal conductivity measured in Watts / (meter·°K)
(These units work because a Watt is a Joule/second.)

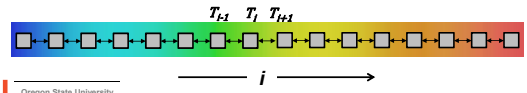
In plain words, this all means that temperatures, left to themselves, try to even out. The greater the temperature differential, the faster the evening-out process goes.

Numerical Methods: Changing a Derivative into Discrete Arithmetic

11

$$\frac{\partial T}{\partial t} = \frac{T_{t+\Delta t} - T_t}{\Delta t}$$

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i-1} - 2T_i + T_{i+1}}{(\Delta x)^2}$$



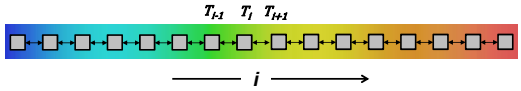
Multicore Block Data Decomposition: 1D Heat Transfer Example

12

$$\rho C \frac{\partial T}{\partial t} = k(\frac{\partial^2 T}{\partial x^2})$$

$$\downarrow$$

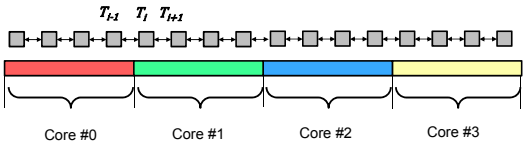
$$\frac{\Delta T}{\Delta t} = \frac{k}{\rho C}(\frac{\Delta^2 T}{\Delta x^2}) \longrightarrow \Delta T_i = \left(\frac{k}{\rho C}\right)\left(\frac{T_{i-1} - 2T_i + T_{i+1}}{(\Delta x)^2}\right)\Delta t$$



As a side note: the quantity ***k/(ρC)*** has the unlikely units of m^2/sec!

1D Data Decomposition: Partitioning Strategies

13



Should you allocate the data as one large global-memory block (i.e., shared)?

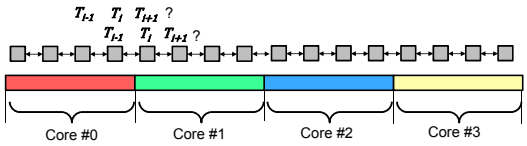
Or, should you allocate it as separate arrays, each dedicated to its own core?

Does it matter?

.

Allocate as One Large Continuous Global or Malloc'ed Array

14



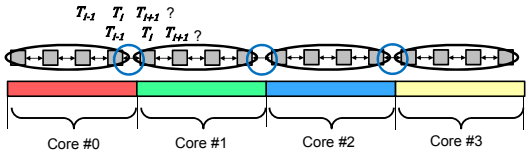

```
float  Temps[ARRAYSIZE];
float *Temps = (float *)malloc( ARRAYSIZE*sizeof(float) );
float *Temps = new float[ ARRAYSIZE ];
<< allocate a new[ ] array the same way >>

omp_set_num_threads( 4 );
for( int t = 0; t < NUM_TIME_STEPS; t++ )
{
        #pragma omp parallel for default(none), shared(Temperatures)
        for( int i = 1; i < ARRAYSIZE-1; i++ )
        {
                << compute ΔTi using Ti-1, Ti, and Ti+1 >>
                new[ i ] = Temps[ i ] + ΔTi;
        }
        << copy the new[ ] array to the Temps[ ] array >>
}
```

Pick one way of allocating global or heap data

What happens when computing at the boundaries? Two cores are accessing the same cache line. **False Sharing!**

Allocate as Separate Sub-arrays

15



We could make each sub-array as a thread-local (i.e., private) variable. This would put each sub-array on each thread's individual stack. But, let's not do that just in case these arrays might be large enough to overflow the stack. Although, if we did, it wouldn't change this story.

Be sure to start each sub-array on its own cache line boundary. (See cache notes.)

But, now when we

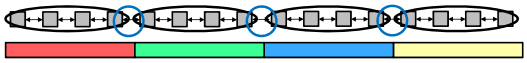
<< compute ΔT_i using T_{i-1}, T_i, and T_{i+1} >>

at the boundaries, T_{i-1} or T_{i+1} might be in another sub-array.

So, we need some logic to reach into the other sub-array to get the adjacent temperature. It is no longer as easy as saying Temps[i-1] or Temps[i+1]. .

1D Compute-to-Communicate Ratio

16



Intracore computing

Intercore communication

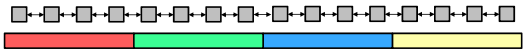
Compute : Communicate ratio = N : 2

where N is the number of compute cells per core

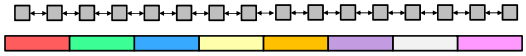
In the above drawing, Compute : Communicate is 4 : 2

How do more Cores Interact with the Compute-to-Communicate Ratio?

17



In this case, with 4 cores, Compute : Communicate = 4 : 2



In this case, with 8 cores, Compute : Communicate = 2 : 2

Think if it as a *Goldilocks and the Three Bears* sort of thing. :-)

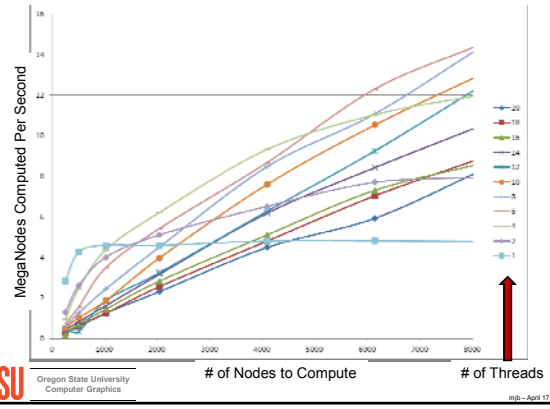
Too little *Compute : Communicate* and you are spending all your time sharing data values across threads and doing too little computing

Too much *Compute : Communicate* and you are not spreading out your problem among enough threads to get good parallelism.

It's difficult to find the "sweet spot" without running experiments

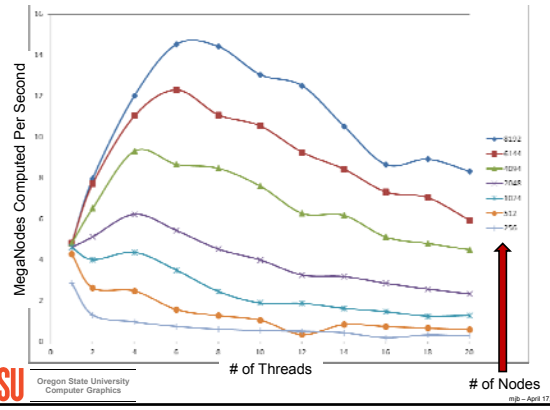
Performance as a Function of Number of Nodes

18



Performance as a Function of Number of Threads

19



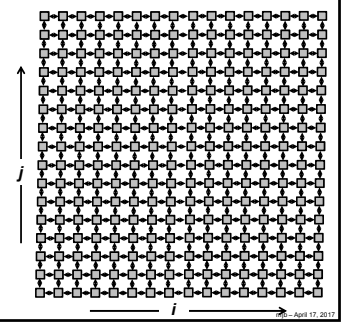
2D Heat Transfer Equation

20

$$\rho C \frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right)$$

$$\frac{\Delta T}{\Delta t} = \frac{k}{\rho C}\left(\frac{\Delta^2 T}{\Delta x^2} + \frac{\Delta^2 T}{\Delta y^2}\right)$$

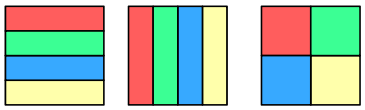
$$\Delta T_{i,j} = \left(\frac{k}{\rho C}\right)\left(\frac{T_{i-1,j} - 2T_{i,j} + T_{i+1,j}}{(\Delta x)^2} + \frac{T_{i,j-1} - 2T_{i,j} + T_{i,j+1}}{(\Delta y)^2}\right)\Delta t$$



2D Domain (Data) Decomposition

21

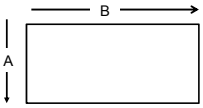
In addition to the issues of size of the compute block, you also have issues of direction.



Direction Issue: Decomposition Order Matters (think cache)

22

float Array[A][B];



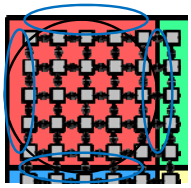
In 2D problems, this is often (but not always) thought of as:

float Array[NY][NX];

0	0
0	1
0	2
0	3
0	...
0	$B-1$
1	0
1	1
1	2
1	3
1	...
1	$B-1$
.	. .
$A-1$	0
$A-1$	1
$A-1$	2
$A-1$	3
$A-1$	...
$A-1$	$B-1$

2D Compute-to-Communicate Ratio

23



Intracore computing

Intercore communication

Compute : Communicate ratio = N^2 : 4N = N : 4

where N is the dimension of compute nodes per core

The 2D Compute : Communicate ratio is sometimes referred to as *Area-to-Perimeter*

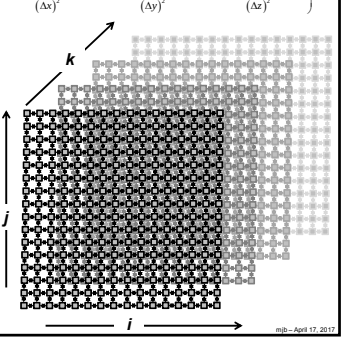
3D Heat Transfer Equation

24

$$\rho C \frac{\partial T}{\partial t} = k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right)$$

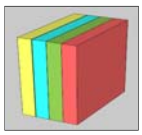
$$\frac{\Delta T}{\Delta t} = \frac{k}{\rho C}\left(\frac{\Delta^2 T}{\Delta x^2} + \frac{\Delta^2 T}{\Delta y^2} + \frac{\Delta^2 T}{\Delta z^2}\right)$$

$$\Delta T_{i,j,k} = \left(\frac{k}{\rho C}\right)\left(\frac{T_{i-1,j,k} - 2T_{i,j,k} + T_{i+1,j,k}}{(\Delta x)^2} + \frac{T_{i,j-1,k} - 2T_{i,j,k} + T_{i,j+1,k}}{(\Delta y)^2} + \frac{T_{i,j,k-1} - 2T_{i,j,k} + T_{i,j,k+1}}{(\Delta z)^2}\right)\Delta t$$

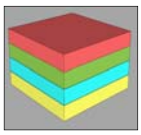


3D Domain (Data) Decomposition

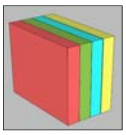
25



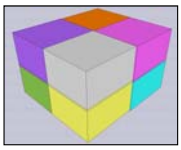
3D Block, *, *



3D *,Block, *



3D *,*,Block

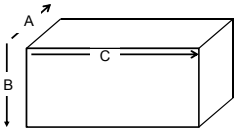


3D Block, Block, Block

Direction Issue: Decomposition Order Matters (think cache)

26

float Array[A][B][C];



In 3D problems, this is often (but not always) thought of as:

float Array[NZ][NY][NX];

3D Compute-to-Communicate Ratio

27

Compute : Communicate ratio = $N^3 : 6N^2 = N : 6$

where N is the dimension of compute nodes per core

In 3D the Compute : Communicate ratio is sometimes referred to as *Volume-to-Surface*