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:

- \( \rho \) is the density in kg/m\(^3\)
- \( C \) is the specific heat capacity measured in Joules / (kg \cdot °K)
- \( k \) is the coefficient of thermal conductivity measured in Watts / (meter \cdot °K) = units of Joules/(meter\cdot sec\cdot °K)

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

As a side note: the quantity \( k/(\rho C) \) has the unlikely units of m\(^2\)/sec!
On a shared memory multicore system, the obvious approach is to allocate the data as one large global-memory block (i.e., shared).

You will actually need two such arrays, one to hold the current temperature values that you are reading from and one to hold the next temperature values that you are writing to.

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>

#define NUM_TIME_STEPS          100

#ifndef NUMN
#define NUMN                                  16 // total number of nodes
#endif

#ifndef NUMT
#define NUMT                                     4 // number of threads to use
#endif

#define NUM_NODES_PER_THREAD    ( NUMN / NUMT )

float              Temps[2][NUMN];
int Now;            // which array is the "current values"= 0 or 1
int Next;            // which array is being filled = 1 or 0

void                   DoAllWork( int );
```

Allocate as One Large Continuous Global Array

```c
#include <stdio.h>
#include <math.h>
#include <omp.h>

#define NUM_TIME_STEPS          100

#ifndef NUMN
#define NUMN                                  16 // total number of nodes
#endif

#ifndef NUMT
#define NUMT                                     4 // number of threads to use
#endif

#define NUM_NODES_PER_THREAD    ( NUMN / NUMT )

float              Temps[2][NUMN];
int Now;            // which array is the "current values"= 0 or 1
int Next;            // which array is being filled = 1 or 0

void                   DoAllWork( int );
```

What happens if two cores are writing to the same cache line?
False Sharing!
What happens if two cores are writing to the same cache line? False Sharing!

Allocate as Separate Thread-Local (private) Sub-arrays

Allocate as Separate Thread-Global-Heap Sub-arrays
Allocate as Separate Thread-Global-Heap Sub-arrays

```c
float *nextTemps = new float[NUM_NODES_PER_THREAD];
for (int i = 0; i < NUM_NODES_PER_THREAD; i++)
    nextTemps[i] = Temps[first+i];
...

// read from Temps[], write into nextTemps[]
for (int steps = 0; steps < NUM_TIME_STEPS; steps++)
{
    // all the other nodes in between:
    for (int i = 1; i < NUM_NODES_PER_THREAD-1; i++)
    {
        float dtemp = ((K / (RHO*C)) * 
            (Temps[first+i-1] - 2.*Temps[first+i] + Temps[first+i+1]) / 
            (DELTA*DELTA) ) * DT;
        nextTemps[i] = Temps[first+i] + dtemp;
    }
    ...
    // don't update the global Temps[] until they are no longer being used:
    #pragma omp barrier

    // update the global Temps[]:
    for (int i = 1; i < NUM_NODES_PER_THREAD-1; i++)
    {
        Temps[first+i] = nextTemps[i];
    }
    #pragma omp barrier
    // be sure all global Temps[] are updated:
}
```

---

1D Compute-to-Communicate Ratio

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

Where N is the number of compute cells per core

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

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

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

Think of 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

Performance as a Function of Number of Threads

2D Heat Transfer Equation

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

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

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

2D Domain (Data) Decomposition

In addition to the issues of size of the compute block, you also have issues of direction.
Direction Issue: Decomposition Order Matters (think cache)

```c
float Array[A][B];
```

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

```c
float Array[NY][NX];
```

2D Compute-to-Communicate Ratio

\[
\text{Compute} : \text{Communicate ratio} = \frac{N^2}{4N} = \frac{N}{4}
\]

where \(N\) is the dimension of compute nodes per core

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

Direction Issue: Decomposition Order Matters (think cache)

3D Heat Transfer Equation

\[
\rho C \frac{\partial T}{\partial t} = \nabla \cdot \left( k \nabla T \right) + \Delta T
\]

3D Domain (Data) Decomposition

3D Heat Transfer Equation

\[
\rho C \frac{\partial T}{\partial t} = \nabla \cdot \left( k \nabla T \right) + \Delta T
\]

3D Domain (Data) Decomposition

3D Block, Block, Block

3D Block, *, *

3D *, Block, *

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

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

Direction Issue: Decomposition Order Matters (think cache)

3D Compute-to-Communicate Ratio

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*