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:

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

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

#define NUMN                                  1024 // total number of nodes
#define NUMT                                     4 // number of threads to use
#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 );
```

```c
void                   DoAllWork( int me )
{     // what range of the global Temps array this thread is responsible for:
    int first = me * NUM_NODES_PER_THREAD;
    int last  = first + ( NUM_NODES_PER_THREAD  - 1 );
    for( int step = 0; step < NUM_TIME_STEPS; step++ )
    {
        float left = 0.;
        if( me != 0 )
            left = Temps[Now][first-1];
        float dtemp = ( ( K / (RHO*C) ) *
                         ( left - 2.*Temps[Now][first] +  Temps[Now][first+1] ) / ( DELTA*DELTA ) ) * DT;
        Temps[Next][first] = Temps[Now][first] + dtemp;

        for( int i = first+1; i <= last-1; i++ )
            float dtemp = ( ( K / (RHO*C) ) *
                             ( Temps[Now][i-1] - 2.*Temps[Now][ i ] + Temps[Now][i+1] ) / ( DELTA*DELTA ) ) * DT;
        Temps[Next][ i ] = Temps[Now][ i ] + dtemp;
    }
}
```
DoAllWork(), II

```c
// last element on the right:
{
    float right = 0.;
    if( me != NUMT-1 )
        right = Temps[Now][last+1];
    float dtemp = ( ( K / (RHO*C) ) * 
                    ( Temps[Now][last-1] - 2.*Temps[Now][last] + right ) / ( DELTA*DELTA ) ) * DT;
    Temps[Next][last] = Temps[Now][last] + dtemp;
}
```

// all threads need to wait here so that all Temps[Next]["" values are filled:
#pragma omp barrier

// want just one thread swapping the definitions of Now and Next:
#pragma omp single
{
    Now = Next;
    Next = 1 - Next;
}  // implied barrier exists here:
```
2D Heat Transfer Equation

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

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

3D Heat Transfer Equation

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

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