Data Decomposition

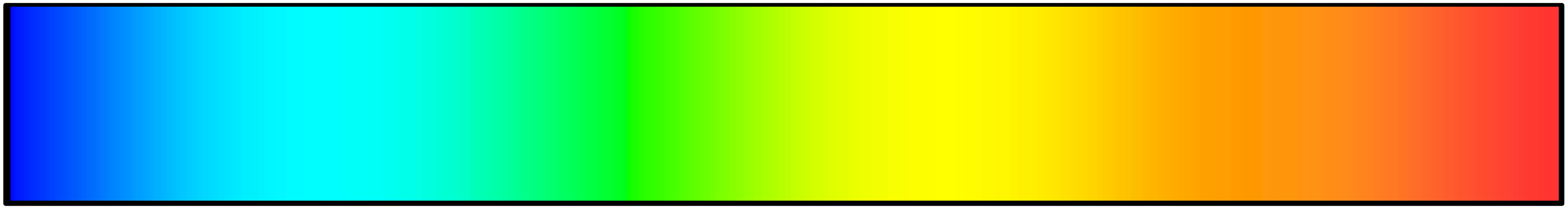
Oregon State University

Mike Bailey

mjb@cs.oregonstate.edu

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

Multicore Block Data Decomposition: 1D Heat Transfer Example



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:

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)

= units of Joules/(meter· sec · °K)

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

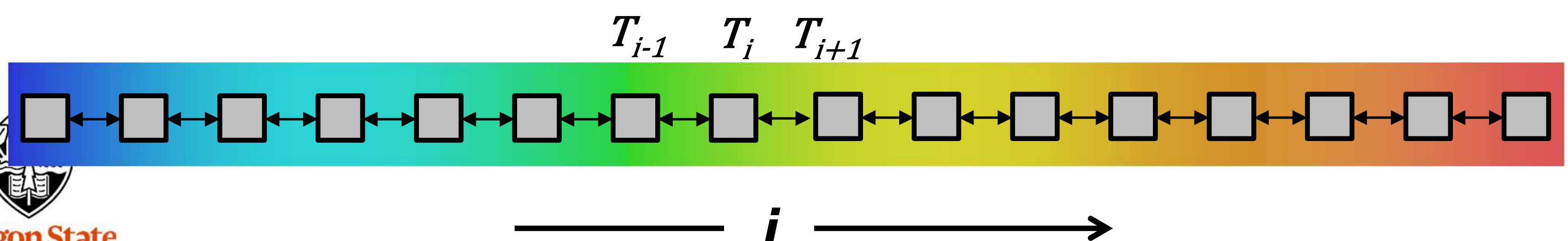
Numerical Methods: Changing a Derivative into Discrete Arithmetic

How fast the temperature is changing within the bar

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

How much the temperature changes over time

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



Multicore Block Data Decomposition: 1D Heat Transfer Example

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

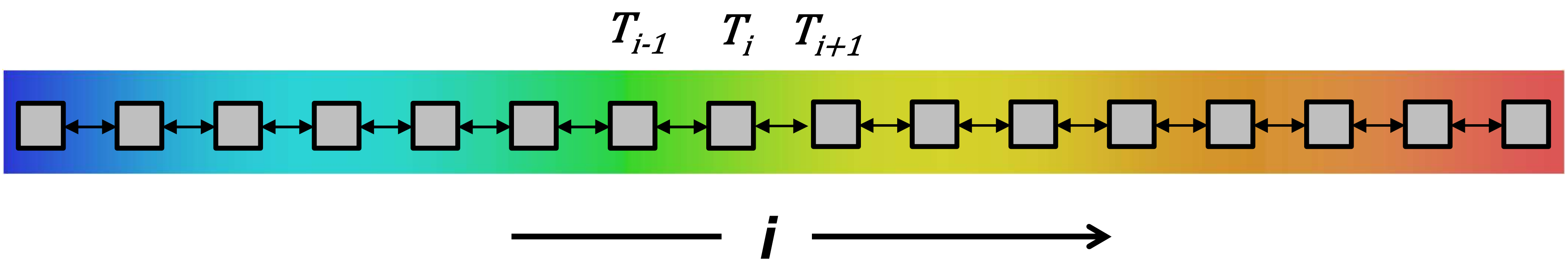
$$\frac{\Delta T}{\Delta t} = \frac{k}{\rho C}\left(\frac{\Delta^2 T}{\Delta x^2}\right)$$

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

How much the temperature changes in the time step

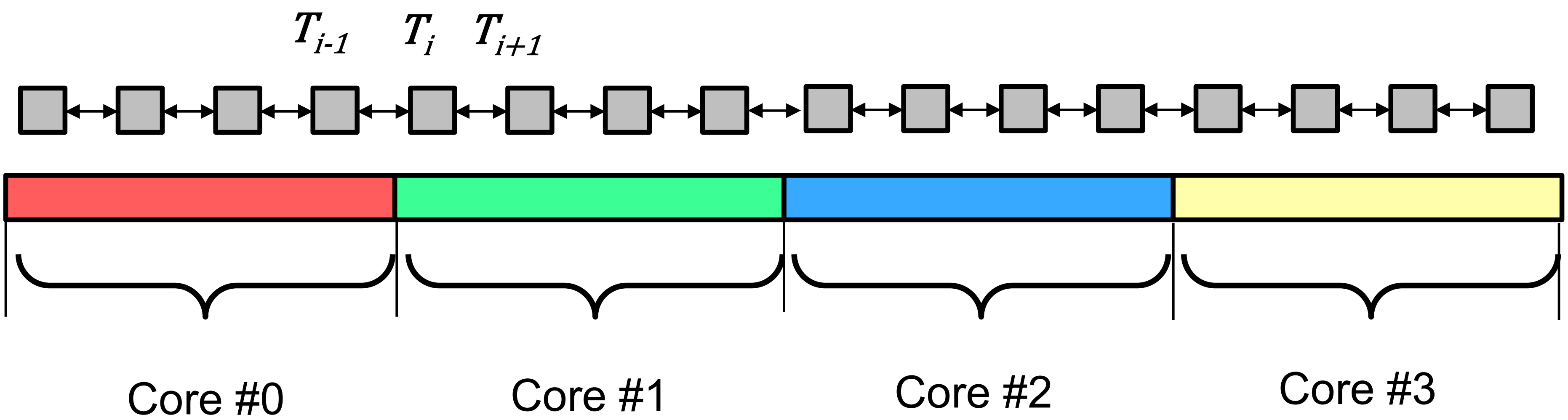
Physical properties of the material

How fast the temperature is changing within the bar



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

1D Data Decomposition: Partitioning Strategies



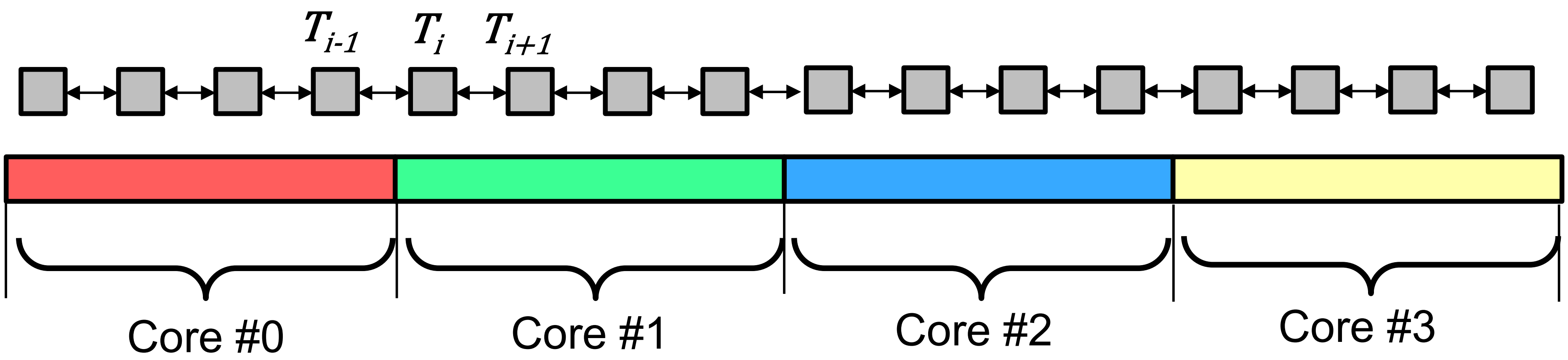
On a shared memory multicore system, the obvious approach is to allocate the data as one large global-memory block (i.e., shared).

You 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.

1D Data Decomposition: Partitioning

```
#include <stdio.h>
#include <math.h>
#include <omp.h>
#define NUM_TIME_STEPS          100

#ifndef NUMN
#define NUMN                            1024        // 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




```
omp_set_num_threads( NUMT );
Now  = 0;
Next  = 1;

for( int i = 0; i < NUMN; i++ )
        Temps[Now][ i ] =   0.;
Temps[Now][NUMN/2] = 100.;

double time0 = omp_get_wtime( );

#pragma omp parallel default(none) shared(Temps,Now,Next)
{
          int me = omp_get_thread_num(  );
         DoAllWork( me );                              // each thread calls this
}

double time1 = omp_get_wtime( );
double usecs = 1000000. * ( time1 - time0 );
double megaNodesPerSecond = (float)NUM_TIME_STEPS * (float)NUMN / usecs;
```

DoAllWork( ), I

```
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++ )
        {
                // first element on the left:
                {
                        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;
                }

                // all the nodes in between:
                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;
                }
```

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

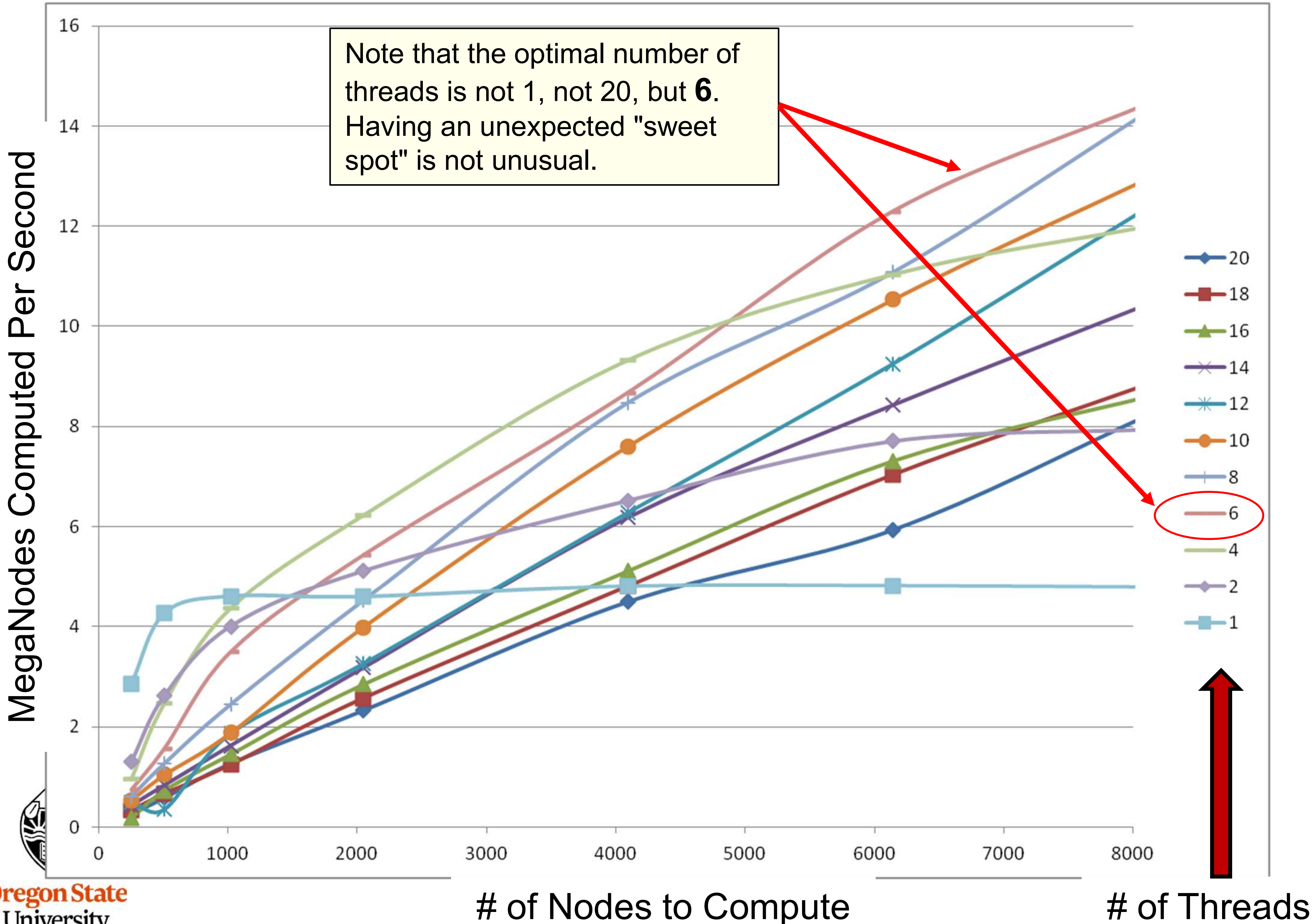
DoAllWork( ), II

```
        // 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:

    }  // for( int step = …
}
```

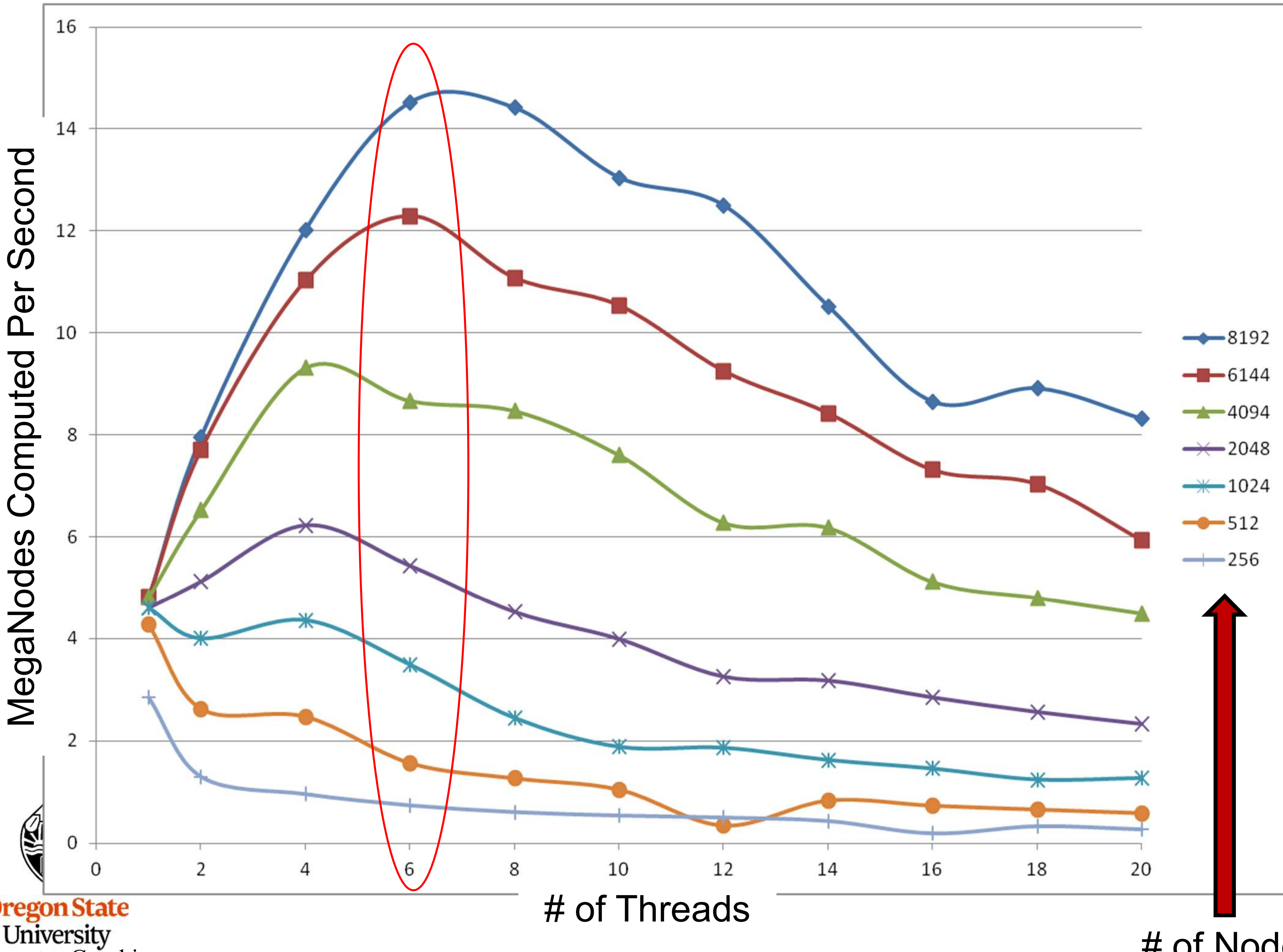
What happens if two cores are writing to the same cache line?
False Sharing!

Because each core is working from left to right across the data, I am guessing that there is little cache line conflict.

Performance as a Function of Number of Nodes



Performance as a Function of Number of Threads



Wait! Why is Peak Performance Happening at 6 Threads, not 1 or 20?

This shows that, for this particular problem, there is a "sweet spot" at ***6 threads***. The logic behind this goes something like this:

- If I am not utilizing enough cores, then I am not bringing enough compute power to bear.
- If I am utilizing too many cores, then each core doesn't have enough to do and too much time is being spent getting values from the memory that another core is computing with.

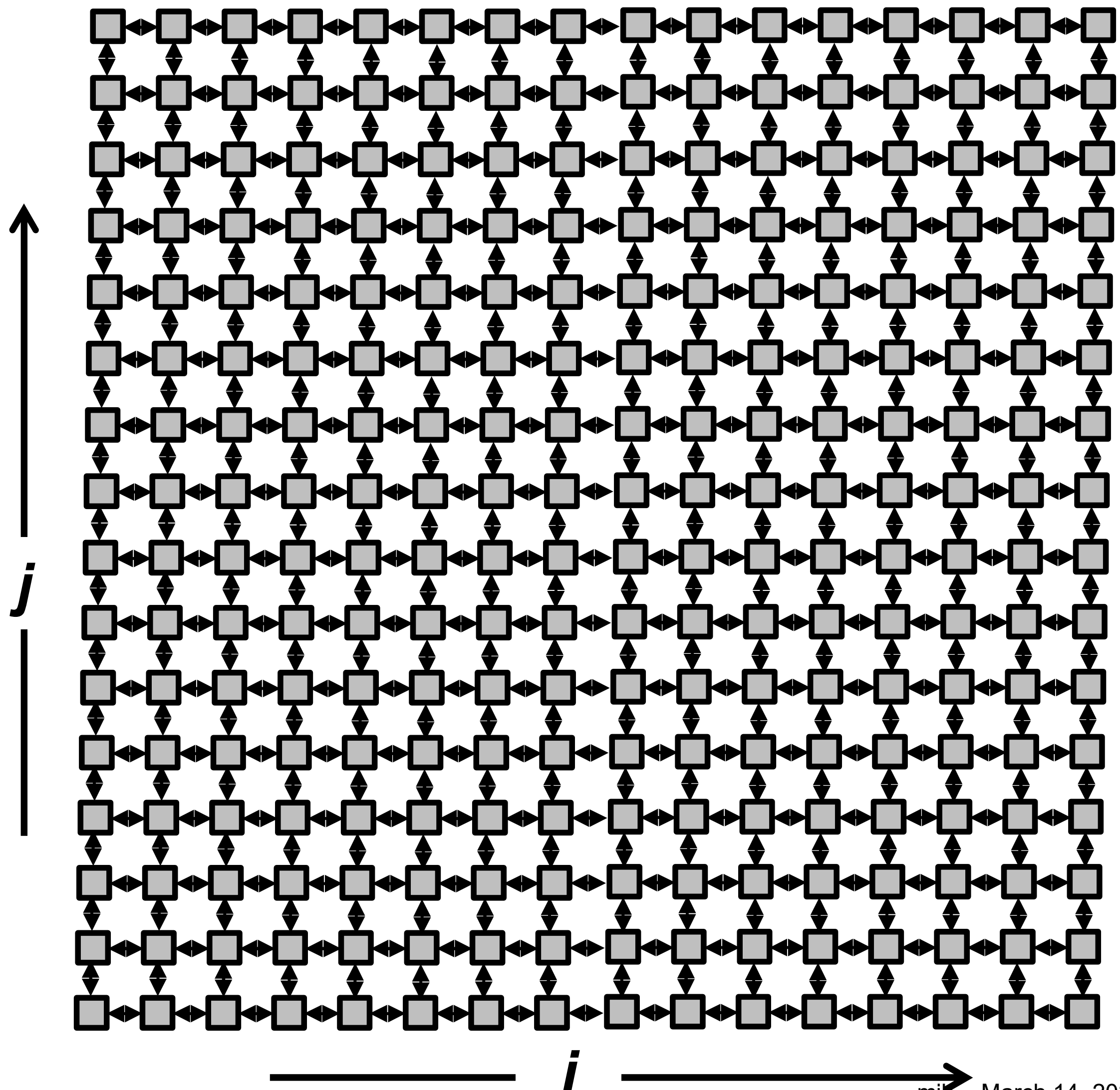
This is known as ***Compute-to-Communicate Ratio*** issue.
This is coming up soon in another noteset.

2D Heat Transfer Equation

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

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

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



3D Heat Transfer Equation

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

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

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