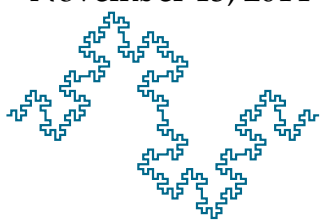
Towards the Scalability and Hybrid Parallelization of a Spatially Variant Lattice Algorithm

Henry R. Moncada

Advisor
Dr. Shirley Moore and Dr. Raymond C. Rumpf

High Performance Computational Science Laboratory (HPCS Lab)
Department of Computational Science
University of Texas at El Paso

November 13, 2014

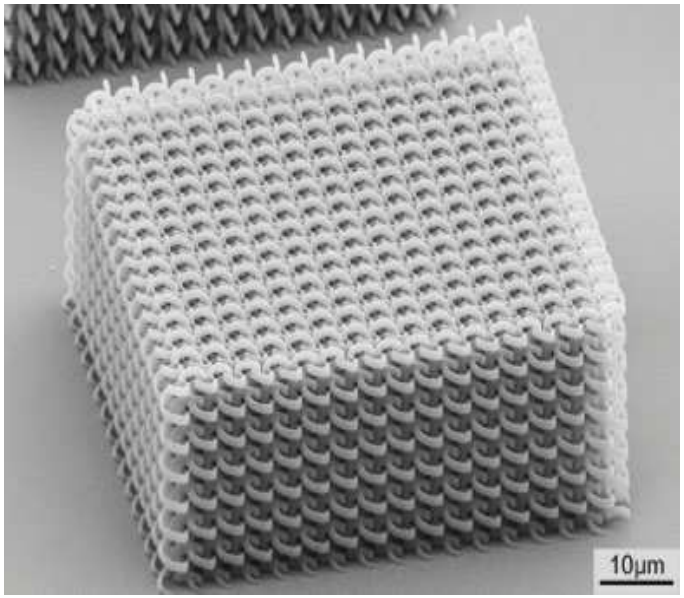


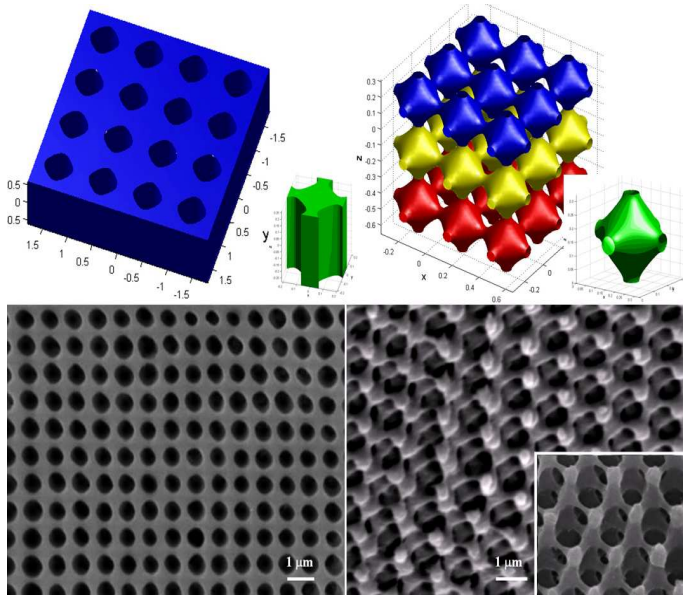
OUTLINE

WHAT IS A METAMATERIAL?

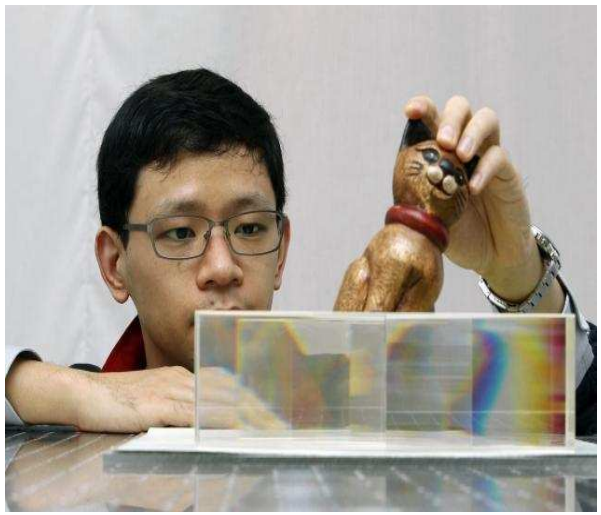
- Metamaterials are **artificial materials** engineered to have properties that have not yet been found in nature.
- They are **assemblies of multiple individual element**s fashioned from conventional materials such as metals or plastics, but the materials are usually constructed into repeating patterns, often with microscopic structures.
- Metamaterials derive their properties not from the compositional **properties of the base materials**, but from their exactingly-**designed structures**.
- Their precise shape, geometry, size, orientation and arrangement can affect waves of light (electromagnetic radiation) or sound in a manner not observed in natural materials.
- These metamaterials achieve desired effects by incorporating structural elements of sub-wavelength sizes, i.e. features that are actually smaller than the wavelength of the waves they affect

CELL STRUCTURES METAMATERIALS



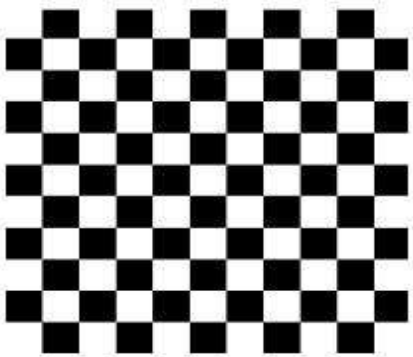


INVISIBILITY CLOAKS (APPLICATION)

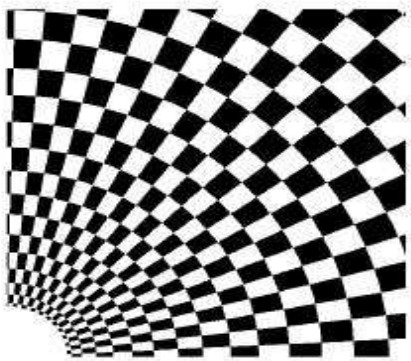


SPATIALLY VARIANT LATTICE

- The method **is not** a coordinate transformation technique.
- It can **more easily reproduce** complicated and arbitrary spatial variance.
- To control how the unit cell orientation, lattice spacing, and fill fraction are to be spatially varied throughout the volume of the lattice, **separate functions must be constructed** that define each of these parameters as a function of position.

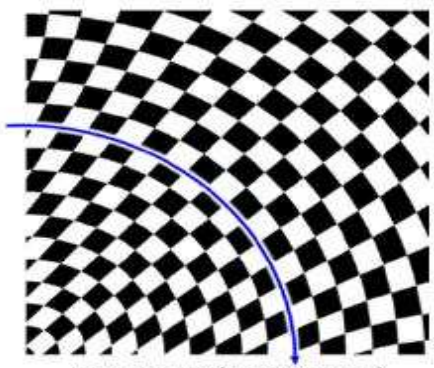


Uniform Lattice



Lattice with Conventional Bend

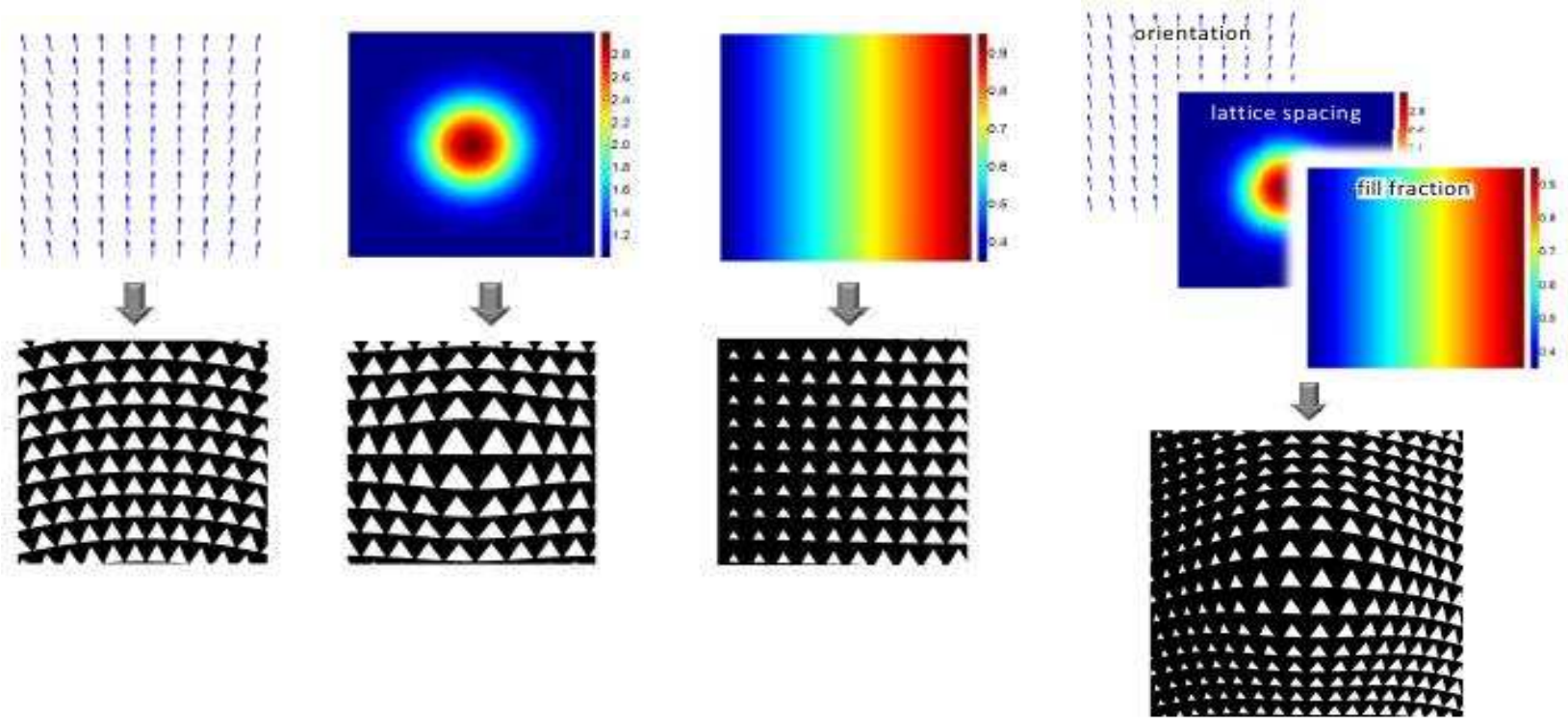
Lattice is:
- Smooth
- Continuous
- Defect free
- Deformations minimized



Lattice with Improved Bend

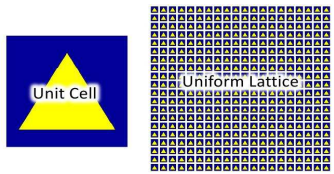
SPATIALLY VARYING LATTICE ATTRIBUTES

. . . at the same time

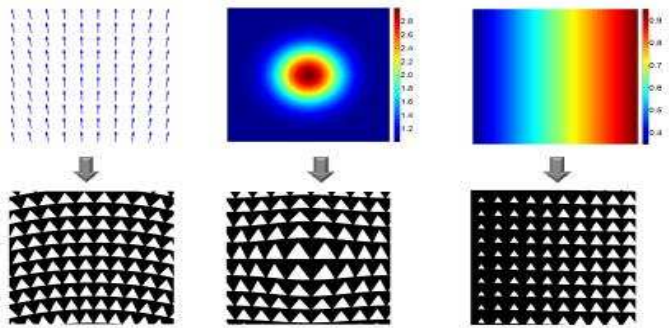


SPATIALLY VARIANT ALGORITHM

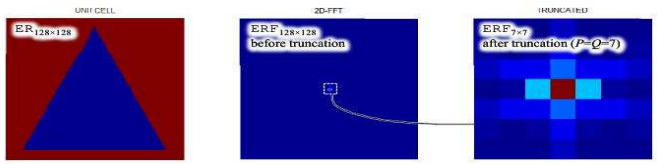
1. The process starts by designing the baseline unit cell that provides whatever properties are desired.



2. Define the spatial variance: Orientation, Lattice spacing, Fill fraction,...

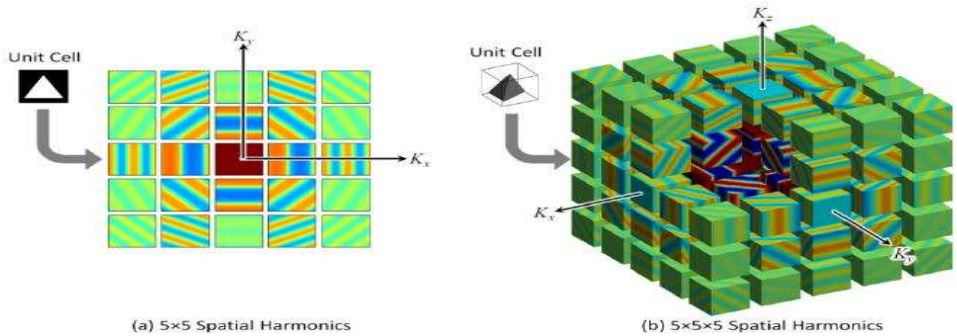


3. Fourier transform of the unit cell.

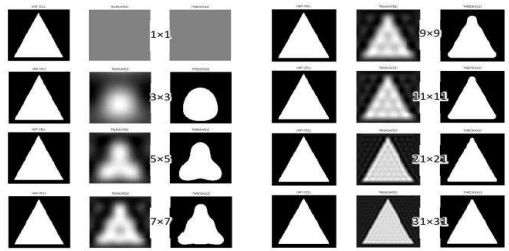


SPATIALLY VARIANT ALGORITHM

4. Truncate Spatial Harmonics and Consequences of Truncating Spatial Harmonics

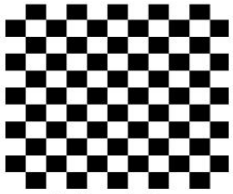


(a) 5×5 Spatial Harmonics (b) 5×5×5 Spatial Harmonics

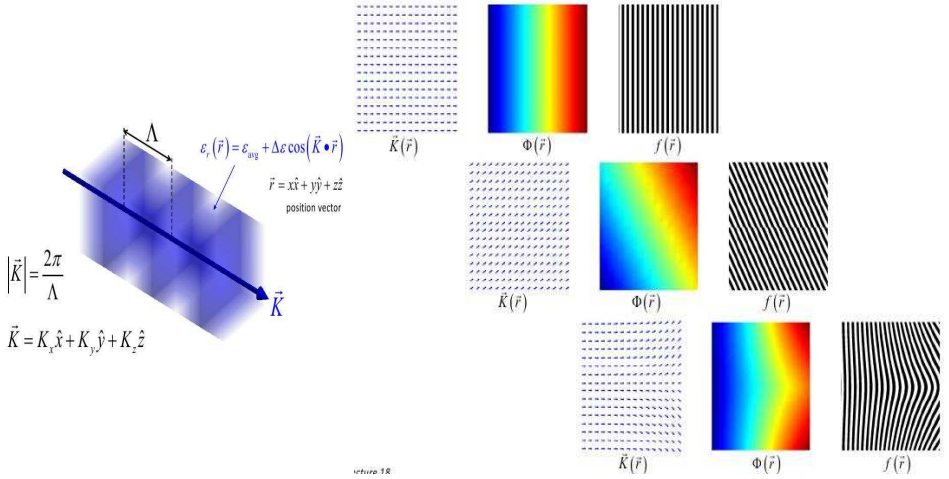


SPATIALLY VARIANT ALGORITHM

5. Initialize the overall lattice to zero.

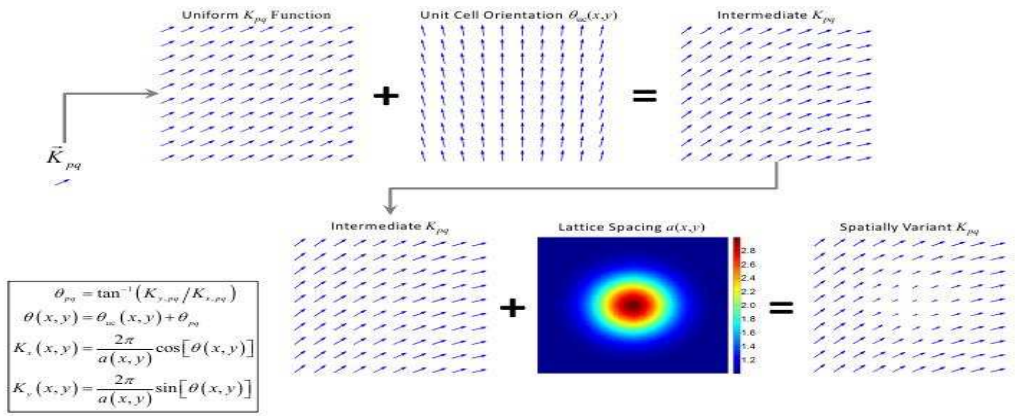


6. Loop over each Spatial Harmonic
   a. Construct a grating vector function $K(r)$ that is uniform across the grid according to the grating vector of the spatial harmonic.



SPATIALLY VARIANT ALGORITHM

b. Spatially **vary the orientation** of $K(r)$ as a function of position according to the direction field.
c. **Scale the magnitude** of $K(r)$ as a function of position to spatially vary the period.
d. **Calculate the grating phase function** $\phi(r)$ from the spatially variant $K(r)$.



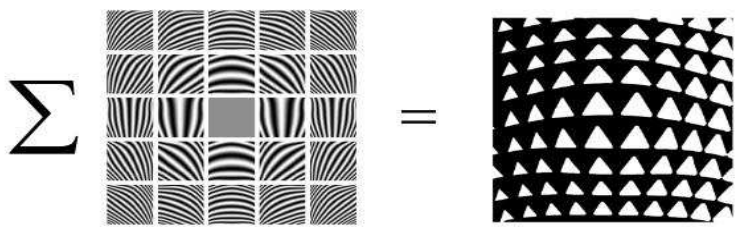
SPATIALLY VARIANT ALGORITHM

e. Construct the spatially variant $1D$ grating.
f. Add the $1D$ grating to the overall lattice. Generate the spatially variant grating for the pq spatial harmonic

$$\varepsilon_{pq}(x,y) = A_{pq} \exp\left(j\Phi_{pq}(x,y)\right)$$

Add this to analog grating

$$\varepsilon_{analog}(x,y) = \varepsilon_{analog}(x,y) + \mathbb{R}(\varepsilon_{pq}(x,y))$$



6. Choose a threshold to get desired fill fraction f. Fill fraction can be spatially varied simply by making γ a function of position.

$$\begin{aligned} \gamma(x,y) &= \cos(\pi f(x,y)) && \text{(For a sinusoidal lattices)} \\ \varepsilon_{binary}(x,y) &= \varepsilon_{analog}(x,y) > \gamma && \text{(Binary grating)} \end{aligned}$$

LOOP ALGEBRA SPATIALLY VARIANT ALGORITHM

```
% Compute the  Derivative Operators
NS = [Nx Ny]; RES = [dx dy]; BC = [1   1];
[DX,D2X,DY,D2Y] = fdder(NS,RES,BC);

LOOP TO SOLVE
    % Spatially variant K-Field
     Kx = KX(nk)*ones(Nx,Ny);
     Ky = KY(nk)*ones(Nx,Ny);
     [TH , Rho] = cart2pol(Kx,Ky);
     TH = TH + THETA;
     [Kx, Ky] = pol2cart(TH, Rho);

    % Gradient Phase Recontruction
     A = [DX; DY];
     b = [Kx(:); Ky(:)];
     PHI = A\b;                    ==> LAPACK (AX=B)
     PHI = reshape(PHI, Nx, Ny);

    % Compute the Spatially Variance 1D Gradient
      S = AMN(nk)*exp(1i*PHI); ==> LAPACK + BLAS (exp(PHI) = D exp(lam

    % Add to unit cell
      UC = UC + S;
%END LOOP
```

Motivation, Why ? and How ?

Our current effort is to develop high performance portable spatially variant codes for parallel architectures.

- The first phase consists of the development of an optimized serial C code as a basis for a high performance Message Passing Interface (MPI) implementation for distributed memory to improve the code's performance.
- In the second phase, the MPI version of the spatially variant algorithm will be produced for portability and evaluation of the benefits
- In the third phase, combining the MPI parallelization with OpenMP shared memory multiprocessing not only benefits us in load balancing but also in cutting down the memory overhead.
- In the fourth phase, we push the limits to speed up our code performance using the advantage of the graphics processing units (GPU) and CUDA (Compute Unified Device Architecture).
- In the fifth phase, We expect to develop other scalable computational electromagnetics algorithms as part of this effort.

What is a High-Performance Computing (HPC)?

- HPC is the use of super computers and parallel processing techniques for solving complex computational problems.
- HPC technology focuses on developing parallel processing algorithms and systems by incorporating both administration and parallel computational techniques.
- HPC systems have the ability to deliver sustained performance through the concurrent use of computing resources.

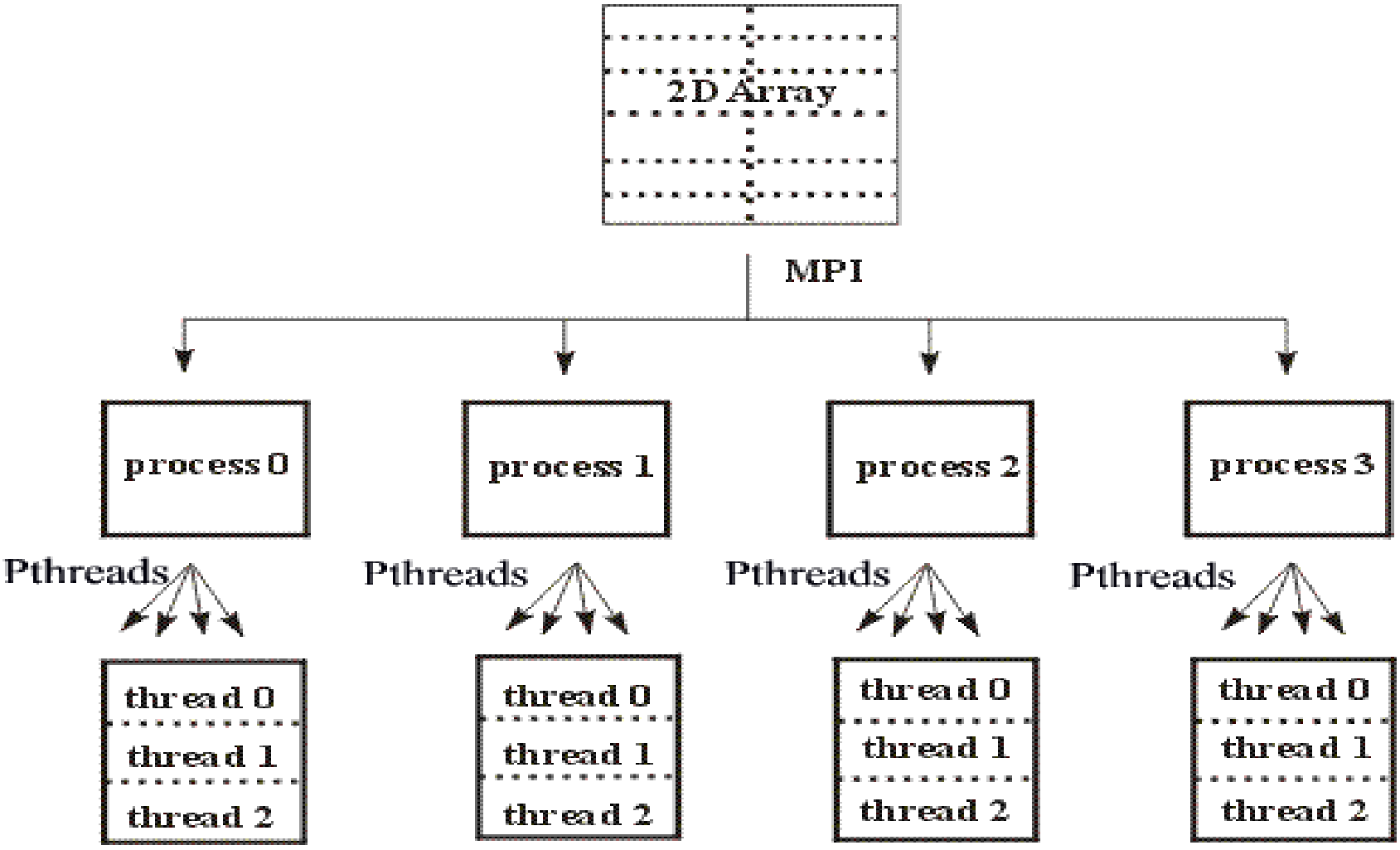
CPU-MULTICORES AND GPUS.

- Single computing component with two or more independent central processing units (called ”cores or CPU”)
- Multicore offers an increase in processing power, but only if the units can be used together efficiently.
- CPU is composed of a only few cores with lots of cache memory that can handle a few software threads at a time.
- Graphics processing unit (GPU) is designed to rapidly manipulate and alter memory to accelerate the creation of images in a frame buffer intended for output to a display.
- GPU-accelerated computing is the use of a graphics processing unit (GPU) together with a CPU to accelerate scientific, engineering, and enterprise applications.
- GPU is composed of hundreds of cores that can handle thousands of threads simultaneously.

MPI and OpenMP

- Message Passing Interface (MPI) is a communication system
- MPI data is moved from the address space of one process to that of another process through cooperative operations on each process.
- OpenMP is a set of compiler directives to express shared Memory Parallelism (MP).
- Used for multi-threaded parallel processing
- Used on shared-memory multi-processor (core) computers
- Part of program is a single thread and part is multi-threaded
- Almost any hardware/OS platform supplies a native MPI and OpenMP version, and public-domain versions are available as well.
- MPI and OpenMP is designed to allow users to create programs that can run efficiently on most parallel architectures.

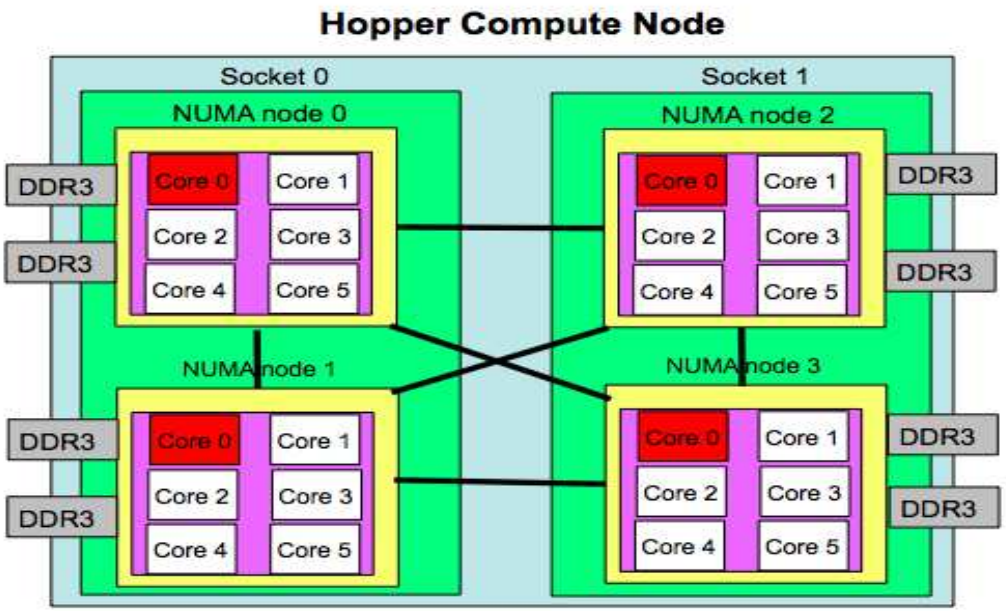
MPI/OpenMP


2D Array
MPI
process 0
process 1
process 2
process 3
Pthreads
Pthreads
Pthreads
Pthreads
thread 0
thread 1
thread 2
thread 0
thread 1
thread 2
thread 0
thread 1
thread 2
thread 0
thread 1
thread 2

Hooper Nersc

- 6,384 nodes
- 2 twelve-core AMD 'MagnyCours' 2.1-GHz processors per node (see die image to the right and schematic below)
- 24 cores per node (153,216 total cores)
- Each core has its own L1 and L2 caches, with 64 KB and 512KB respectively (MPI)
- One 6-MB L3 cache shared between 6 cores on the Magny-Cours processor (OpenMP)

Hooper Nersc (NUMA:Non-Uniform Memory Architecture)




```
#PBS -q debug
#PBS -l mppwidth=24
#PBS -l walltime=00:30:00
#PBS -N my_job
#PBS -e my_job.$PBS_JOBID.err
#PBS -o my_job.$PBS_JOBID.out
#PBS -V

cd $PBS_O_WORKDIR

# Pure MPI
aprun -n 4 -S 1 ./my_prog

# Hybrid MPI/OpenMP
setenv OMP_NUM_THREADS 6
aprun -n 4 -N 4 -S 1 -d 6 ./my_prog
```

Using and Implement LAPACK and BLAS Routines in C

- LAPACK and BLAS subroutines are written in Fortran.
- External libraries can be divided into two parts,
  - How to call the routines in your program
  - How to compile this program
- Difference between **C** and **Fortran**
  - The way matrices are **stored in memory**

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

    - In **C** matrices are stored in row major order. Would be stored as, `a b c d e f g h i`
    - In **Fortran** matrices are stored in column major order. Would be stored as, `a d g b e h c f i`
  - The way arguments are passed to a subroutine.
    - In Fortran the argument is passed by reference.
    - In C, we have to pass pointers to variables instead of the variables themself to the routine.,

LAPACK (Linear Algebra Package)

- CLAPACK routines have names of the form *XYYZZZ*.
  - The first letter *X* indicates the data type the routine expects. *S* Real, *D* Double precision,*C* Complex, *Z* Double complex, the same as double in C
  - The letters *YY* tell the type of matrices the routine deals with. *GE* and *TR*, meaning general and triangular, are the ones we will use.
  - *ZZZ* is the name of the actual computation done by a particular subroutine, e.g., *SV* denotes factor the matrix and solve a system of equations, *LS* denotes solve over or undetermined linear system using orthogonal factorization.
  - For example,
    - **SGEBRD** is a single precision (S) routine that performs a bidiagonal reduction (BRD) of a real general matrix (GE).
    - **DGEEVX** is a double precision (D) routine that computes the eigenvalues and, optionally, the left and/or right eigenvectors (EVX) for real general matrix (GE), X stand for expert

BLAS (Basic Linear Algebra Subprograms)

- Specified set of low-level subroutines that perform common linear algebra operations such as copying, vector scaling, vector dot products, linear combinations, and matrix multiplication
  - Level 1: Contains vector operations on strided arrays: dot products, vector norms, a generalized vector addition of the form

$$\boldsymbol{y} \leftarrow \alpha \boldsymbol{x} + \boldsymbol{y}$$

  - Level 2: TContains matrix-vector operations including a generalized matrix-vector multiplication (GEMV):

$$\boldsymbol{y} \leftarrow \alpha A \boldsymbol{x} + \beta \boldsymbol{y}$$

  - Level 3: Contains matrix-matrix operations, including a "general matrix multiplication" (GEMM), of the form

$$C \leftarrow \alpha AB + \beta C$$

    where A and B can optionally be transposed inside the routine and all three matrices may be strided

Acknowledgement

We would like to acknowledge that this research work was based on work developed by Dr. Raymond C. Rumpf and team lab at EMLab-UTEP. To my advisor Dr. Shirley Moore and Dr. Raymond C. Rumpf for their support and help in accomplishing my research. To the UTEP-Computational Science Program labmates and freinds for their support and regular feedback.

End

QUESTIONS ???