Multi-Domain Pattern

I. Problem

The problem represents computations characterized by an underlying system of mathematical equations, often simulating behaviors of physical objects through discrete time steps. There exists an intuitive algorithm that solves the problem in a single domain (usually spatial domain with discrete coordinates), but that in some cases results in inaccuracy, instability or high algorithmic complexity. Such issues are mediated by using special properties from the underlying equations to mathematically transform the problem space into a different domain that may be able to produce solution in more accurate and efficient manner. Such mathematical transformation must be translated into multi-domain parallel algorithm that exploits computational advantages from multiple domains.

II. Driving Forces

1. **Performance** of the multi-domain algorithm must be superior to the single domain algorithm. This means the overhead of transformation and interaction from the multiple domains must not exceed the computational benefits gained from the newer domain.
2. **Load balancing and data redistribution** must be taken into consideration as the transformation takes place in a domain to another.
3. The **transformation kernel** must be carefully selected and tuned.
4. At each domain, different **data mapping** scheme must be defined.

III. Solution

1. Multiple Domain

   A *domain* provides a logical representation of the input dataset involved in a computation. It gives meaning and logical structure to the dataset given. For example, a spatial domain indicates that the dataset is ordered in physical coordinates in space, whereas a frequency domain views the dataset as a tuple space of wavenumber.

   In general, most of the parallel algorithms represent the problem space in a single domain throughout the lifespan of the computations. The data elements for the problem are distributed and computed with respect to a single domain and persists without having to consider another logical representation. This preserves simplicity of the algorithm and in most cases is optimal both in terms of accuracy and performance.

   For some family of problems, however, it becomes very inefficient or inaccurate to confine the input dataset in a single domain. For example, computing behaviors on a spherical geometrical surface might not be well portrayed using a 2D grid

domain algorithms, but can be improved with additional computations using spherical harmonics domain. Another example can be found in computing the motion of atoms in molecular dynamics. Direct computation of force interaction of all possible two pairs of atoms takes large amount of time in the spatial domain. Close result can be approximated it with surprising accuracy, with appropriate transformation and analysis in the frequency domain. The examples will be discussed in detail in the "Examples" section of this document.

It must be noted, however, that application of multi-domain pattern is highly problem and algorithm specific. The underlying numerical algorithm must exhibit a well-formulated transformation and computational method to produce the solution using a new domain. Depending on many factors such as transformation overhead, convergence rate, and data locality, this pattern may not be suitable in some cases. It is suggested that this pattern is only applied when the computational advantage gained by transforming into a new domain is an order of magnitude greater than the original single domain algorithm.

Multi-domain pattern is an interactive process between two domains. The original domain is not discarded as the new domain in generated. In fact, both domains are involved in computational work, and mutually provide inputs for each other as a series of transformation occurs between the domains. Usually, the original domain is how the user of the program perceives data. The datasets do not have much meaning to the user in the form of newly generated domain. While achieving the goal of accuracy and efficiency in a new domain, it is in many cases necessary to convert the new domain back to the original representation. There can be cascades of more than one domains throughout the lifespan of the program.

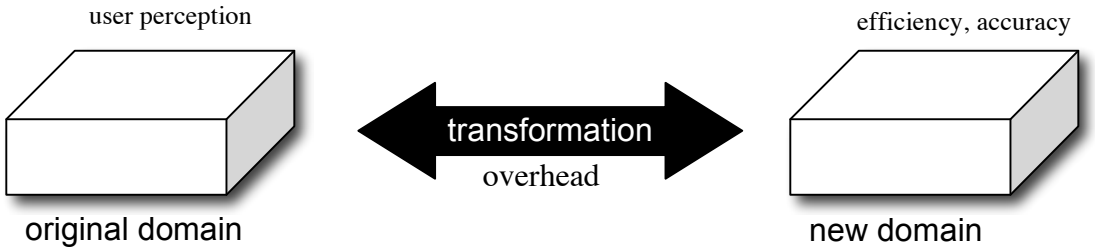

Figure 1. Multiple domains compute and interact at the price of transformation overhead.

2. Transformation Kernel

Switching from one domain to another involves a transformation algorithm. Depending on the nature of the domain, different kernel can be used in the transformation process. For example, conversion to frequency domain can be achieved through Fast Fourier Transform(FFT), whereas spectral domain can be derived using Legendre Transform. The transformation is usually bidirectional, thus it involves inverse transformation when transforming *back* to the original

domain. However, such inverse transformations are often straightforward by inversing the order of computation for the original kernel.

In practice, the transformation kernel is often clearly separated from the rest of the computational code. Other than preserving modularity, such practice makes use of existing transformation kernels developed in the community or vendors. Frequently used parallel transformation kernels such as FFT and Legendre Transform are highly optimized by vendors for wide range of platforms and different environments. By taking advantage of the existing code, the transformation overhead can be reduced significantly.

The computational advantage gained from deriving a new domain must outweigh the overhead of the transformation cost. Also, the parallelization of the kernel must be scalable for any arbitrary sized input. Even if the underlying numerical algorithm has transformation method available, it is the choice of transformation kernel and its parallelization strategy that determines the success of multi-domain algorithm.

For multi-dimensional input dataset, the transformation may need to be separated into multi-phased process. For example, three-dimensional spatial data can be transformed into frequency domain using a series of FFTs. As shown in figure 2, 3D space is distributed in its first dimension and applied 2D FFT at each partition. Each partition is distributed again at the second dimension, and performed 1D FFT for final conversion to frequency domain. Breaking up the transformation into simpler multiple phases provides modularity at individual phases, and again provides wider choice of existing highly optimized algorithms.

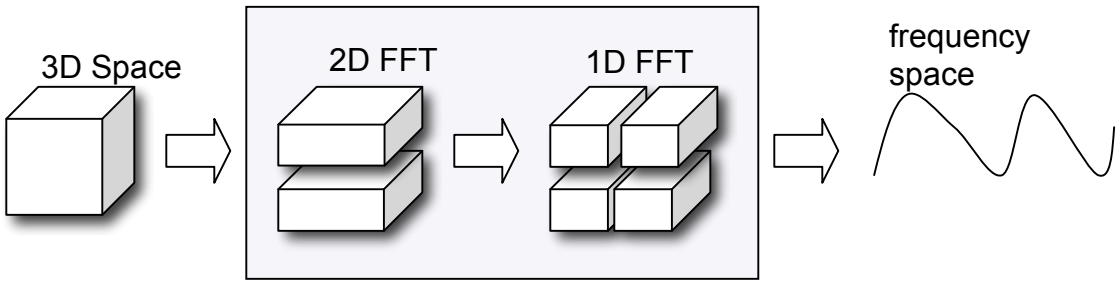

Figure 2. Multi-phased Transformation. Multi-dimensional structure my need to be transformed in multiple phases. Simpler dimensions may indicate wider choice of existing algorithms.

3. Locality / Redistribution / Load Balance

Locality

Multi-domain parallelization involves mapping of datasets to processing units at *each* domain. This means different partition scheme must be used at each domain, and redistribution of data will occur between the transformations as a result. This is simply because data elements local to processor at one domain might not be

continuous or logically proximal at another domain. Using a single partition scheme and using persistent mapping throughout multiple domains will incur data to be scattered without regard to the current domain. Computing the data elements with such mapping will require complex computational algorithms, which are prone to errors and more communication overhead. For this reason, we want to redefine the partition at each domain, so that all the data elements in a partition are continuous and logically intact after the transformations occur.

Redistribution
Since we are using different partition mapping each of the domains, we need to ensure that the local data needs to be redistributed across other processing units. Using existing optimized collective operations, if available, such as scatter-and-gather scheme will help reduce the redistribution cost. In some cases, simply replicating the partitions across group of processors increases parallelism for the computations in the upcoming domain. Fundamentally, the partition needs to be carefully designed so that it will incur less redistribution efforts or computational advantage in the next domain.

Load Balance
Transformation not only produces scattered data elements, but also uneven workloads across processors. For example, a partition from a regular dense spatial data may experience imbalance of work at spectral domain where number of coefficients varies depending on the frequency (This will be explained in more detail at the Example section). If possible, partitioning needs to take into account the workload of each processor in the next domain after the transformation.

As shown above, partitioning strategy plays a *key role* in minimizing the data redistribution and load balancing across multiple domains.

IV. Difficulty

1. Partitioning at one domain needs to consider load balance and redistribution of data in the *next* domain.
2. Careful choice of transformation kernel is needed from collection of many existing developments.

V. Related Patterns

Multi-Level Grid (In a way, each level can be viewed as different domain), LoadBalancing,

VI. Example

N-Body Simulation Using *Particle-Mesh Ewald*
There is a growing interest in the scientific community to develop molecular dynamic simulations to capture the behavior of individual atoms at sub-microscopic level. Such simulations play an important role in commercial field such as drug design, and also heavily used by researchers in the scientific community to study biological systems or material

composition. The simulation involves computing velocity and position coordinates of each atom as they interact with each other in discrete time steps. For N number of particles simulating the interaction between all particles take following form.

$$Force_i = \sum_{j \neq i}^{N} \frac{q_j}{|r_i - r_j|}, \quad \forall\, i = 1,2,3,...N$$

where q is coulomb force of the atom and r is the location of the atom.

In other words, for each of the atom, force exerted by all other atom must be summed up for computing the new position. This takes N(N-1) computations with asymptotic complexity of $O(N^2)$. Even for the parallel computations, the number of processing power needed is prohibitive at this complexity, and achieves poor scalability. Just to increase the number of particles by a magnitude involves increasing the number of processor by a hundred fold.

In order for the simulations to be accurate and meaningful, the molecules need to be placed in a natural environment such as water bath or inside a membrane wall. To simulate such effect, a technique called Periodic Boundary Conditions (PBC) has been devised, where original molecule particles are replicated in multiples in all directions. Each replicated particles simply mimic the motions of the original group of particles (cell), but the original particles are influenced by all particles in the system. This increases the total number of atoms to a significant factor, making the simulation even more difficult.

For periodic replicated cells, however, there is an approximation method that can simulate the system in significantly less time. *Particle Mesh Ewald* method divides the computation of the forces into two fast converging sums as the following.

$$Force_i = Real\text{-}Space(short\text{-}range) + K\text{-}Space(long\text{-}range)$$

The *real-space* atoms within a certain range (cut-off distance) are computed directly using summations, and all other long ranged atom interactions are approximated in the *k-space* frequency domain. We will look into the k-space computation in detail because it involves the transformation into Fourier Domain. The Particle Mesh Ewald algorithm starts with expressing the k-space particles using a Gaussian distribution. Using numerical approximation techniques, the parameters for the distribution is determined and expressed in a reciprocal wave as shown in figure 3.

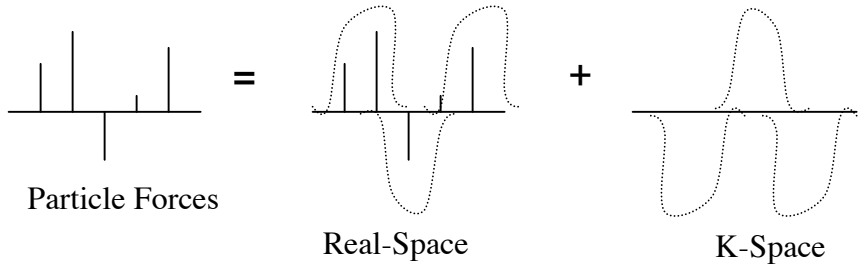

Figure 3. K-Space takes the form of reciprocal Gaussian distribution of the original particle forces.

The parallel computation of the k-space will proceed, as the Gaussian forces are interpolated into a regular 3-Dimensional grid. The 3D grid will be now be transformed into the frequency domain, by performing a multiple phases of Fast Fourier Transforms (FFT). The data will be partitioned in the first dimension, redistributed among the processing units, which will locally perform 2D FFT on the dataset. The dataset will once again be partitioned at the second dimension among the processing units for the final 1-dimensional FFT into the frequency domain. The FFT kernels are naturally parallel and can be computed in O(NlogN).

At the frequency domain we now have a better representation of the periodic replicated cells. The only additional computation needed is to linearly perform a few multiplications using Ewald electrostatic kernel for each of the transformed grids. This finishes in O(N) time.

The rest of the process is a reversal to the previous steps. Inverse FFT is performed on the first dimension, and the grids are reassembled back along its first dimension. This is followed by a backward 2D FFT, which will generate 3D representation of k-space. The computed results will be interpolated back to Gaussian equations and finally combined with the real-space force. The whole process can be summarized in figure 4.

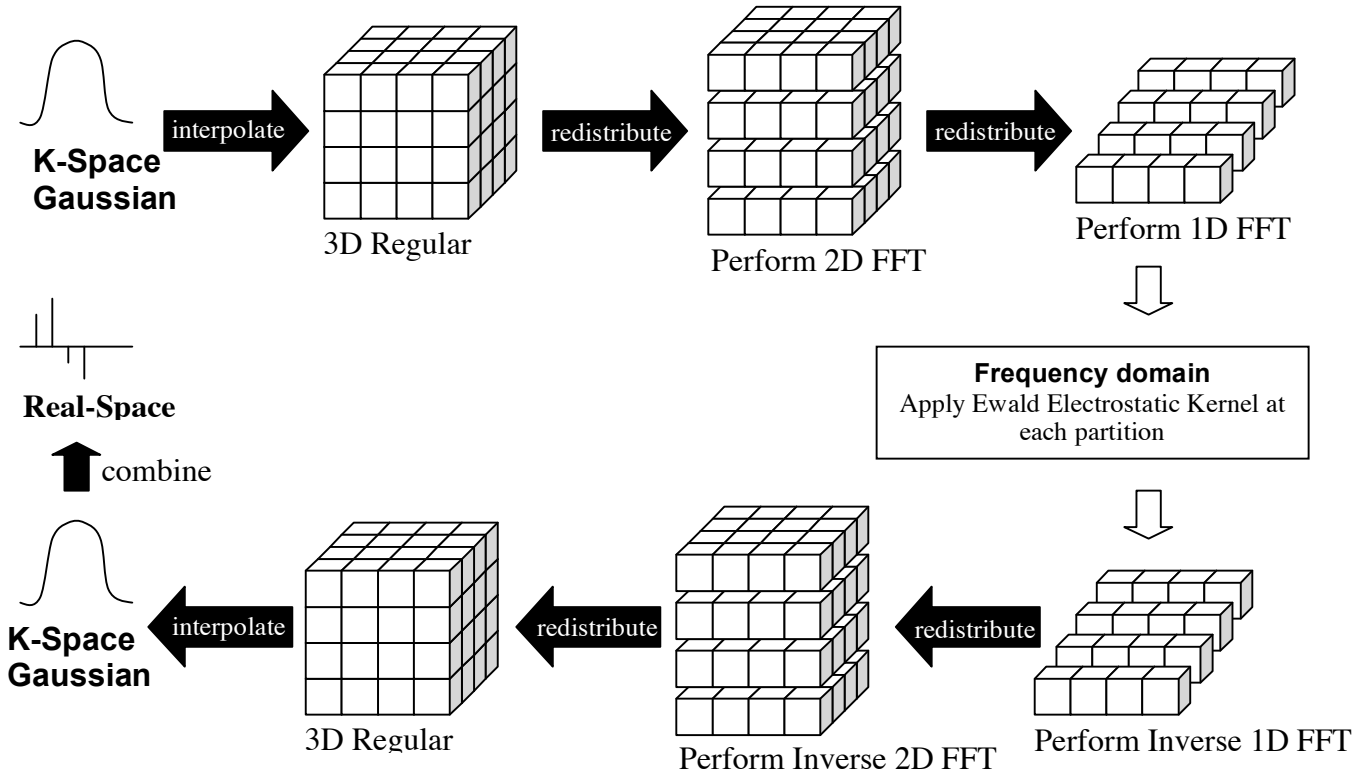

Figure 4. Particle-Mesh Ewald Algorithm. K-Space is transformed in a frequency domain for more efficient computation.

By shifting the workload to the frequency domain, significant amount of pair-wise computational work has been reduced to simple multiplications in linear time. The overhead obtained through the transformation is still cheaper than the original complexity: O(NlogN). Also, FFT is known to scale well under large amount of data. Thus, we claim that the

benefit of the multi-domain algorithm outweighs the transformation overhead. Note not many algorithms feature such transformational methods. To apply multi-domain algorithm, one must ensure similar numerical transformation exists for underlying system of equations.

Shallow Water Model

Computer simulations of the atmospheric circulations are used to predict future weather conditions and to study the mechanisms of global climate change. Widely used atmosphere simulation called Community Climate Model (CCM) features multiple interdisciplinary prediction modules integrated together to simulate different aspects of the atmosphere in discrete time steps. Fluid dynamics of the atmosphere is one of the representative components in the atmosphere model, and is often computed by using shallow water equations.

Intuitively, one can discretize the Earth surface into latitude-longitude grids, and apply 2D Navier-Stokes equations in finite difference method to capture the atmospheric motion. However, this creates two problems for the simulation. First, there is a numerical stability criterion for the grids, namely $\Delta t = \Delta x / v$, where v is the maximum wind velocity. Such criterion restricts the time steps to be extremely small for a significant number of grids. Small time steps will generate prohibitive burden for the processing units. The second problem comes from the fact that the grid space cannot capture the spherical Earth surface accurately. The grids become convergent near the poles, thus the accuracy of computation is seriously impaired.

Spectral domain is an alternative representation to model a spherical surface. In spectral domain, the behaviors on the spherical surface are represented by combination of *spherical harmonic functions*, which are usually orthonormal Legendre polynomials. Spectral domain tends to capture the motions on spherical object more accurately and with less restriction on the time steps. In practice, most of the climate modeling simulations rely on the spectral domain for more accurate and efficient computation.

Transformation to the spectral domain takes multiple phases. The latitude-longitude spatial domain must be first transformed into frequency domain using Fast Fourier Transform (FFT), now containing the latitude-wavenumber pair. From the frequency domain, the spectral domain can be derived by using Legendre transform, which contains coefficients of polynomial for each of the wavenumbers. The inverse of each transform occurs when reversing in direction.

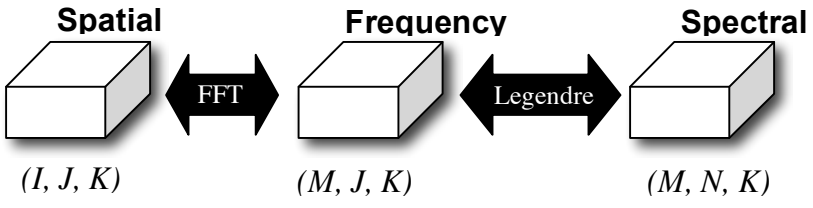

Figure 5. Multiple transformation phases to reach the spectral domain. FFT and Legendre transforms are used between the domains. I=longitude, J=latitude, K=vertical, M=wavenumber, and N=Legendre coefficients.

It is takes two distinct transformation to go from spatial to frequency, and frequency to spectral domain. As shown in figure 5, three-dimensional grid (I,J,K), with I corresponding to longitude, J to latitude, and K to the vertical, is transformed into (M,J,K) space where M represents the wavenumber. Another transformation follows and the spectral domain takes the form of (M,N,K), where N is the coefficients for the Legendre polynomials.

The shallow water equations at the spatial domain contain differential equations representing velocity, momentum, and geo-potential. This can be represented as spherical harmonics equations in the spectral domain with additional terms such as vorticity and divergence. Computations are performed in both domains. The spatial domains computes some terms that it can compute instantly, where as spectral domain computes the sensitive terms related to the spherical surface. Spectral domain's representation helps the computation to be more efficient and also accurate.

Parallelizing the transformation process requires attention in data distribution to ensure maximum data locality and minimal load balancing effort. CCM uses different data mapping for each of the domain to ensure that each contains best locality and balanced workload possible. Figure 6 shows the original data mapping for spatial domain. We assume 4x4 physical grid of processors, and longitude-latitude surface area is block partitioned for each of the processors. Each processor is responsible two symmetric partitions (northern hemisphere and southern hemisphere) along the middle horizon. This assignment allows symmetry of the associated Legendre polynomials to be exploited during the Legendre Transformation. In other words, it promotes more data locality in the upcoming domain.

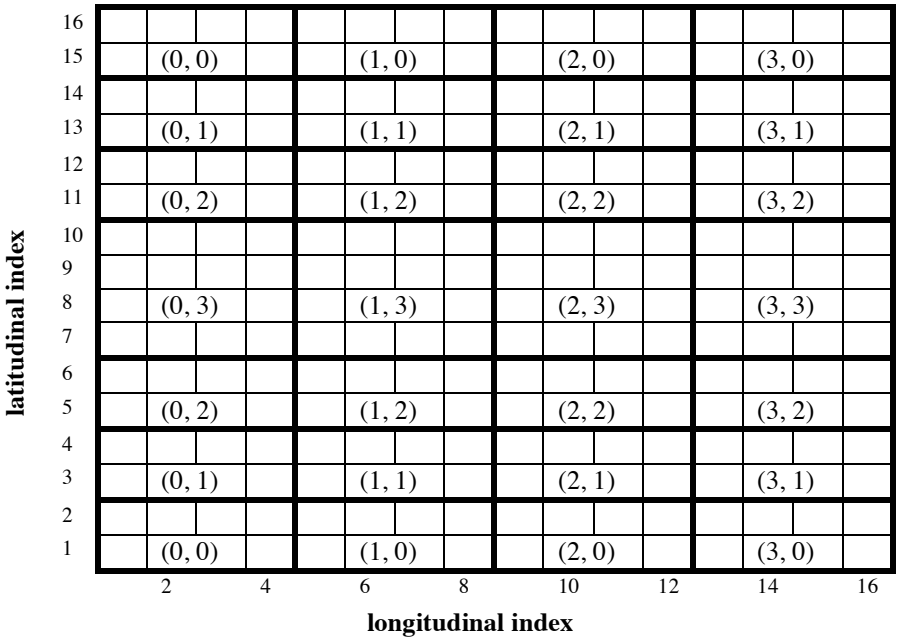

figure 6. Data distribution at the spatial domain.

Physical space is now ready to be transformed into frequency domain. Using existing parallel FFT the spatial domain is converted to the frequency domain. FFT requires communication such as scatter-and-gather between the processors to exchange frequency

information along the latitudinal index. After or even during the transformation phase, data is distributed as shown in figure 7. Latitude-wavenumber dataset is again block-partitioned among the processors, preserving the symmetry along the latitude blocks. Wavenumber space is divided between the columns of processors, but shows a special ordering. This serves to balance the distribution of spectral coefficients in the next spectral domain.

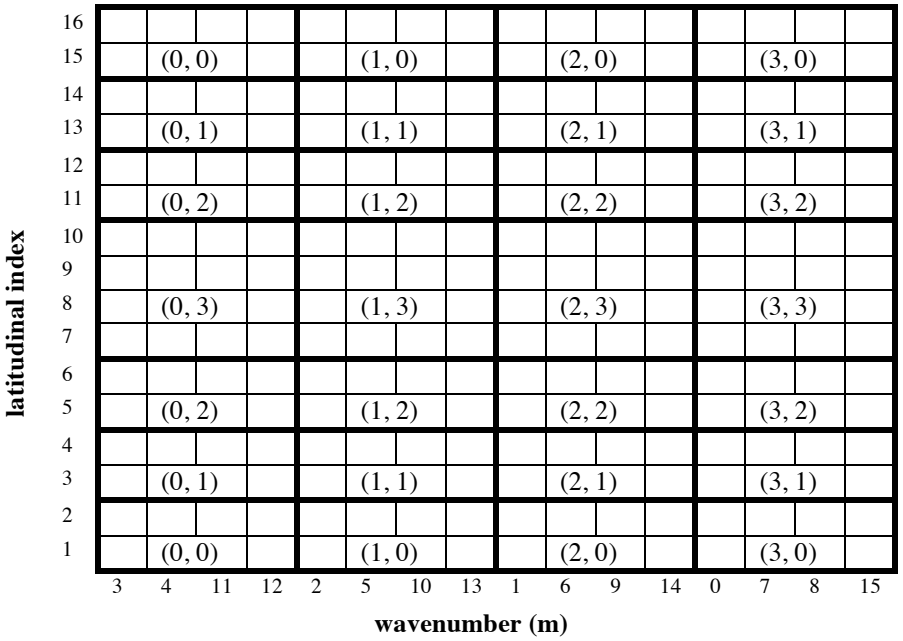

figure 7. Data distribution in frequency domain (after FFT).

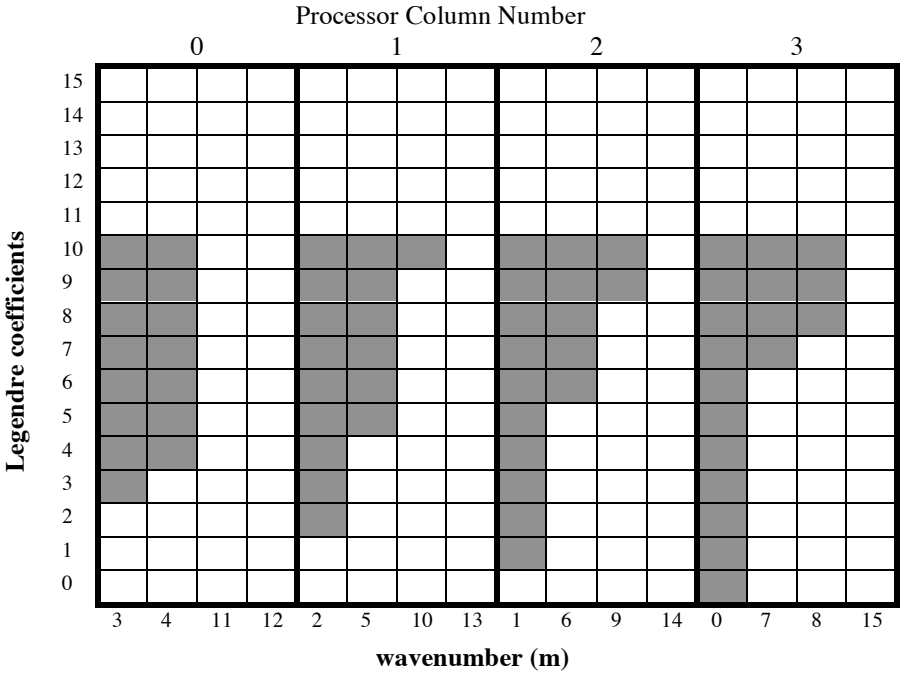

figure 8. Data distribution in spectral domain (after Legendre Transform).

Figure 8 finally shows the mapping in the spectral domain. Parallel Legendre transformation is performed along the latitudinal data, using a scheme similar to the distributed vector sum. Each coefficient depends only on data associated with the same wavenumber. Each processor first computes its own contribution and then broadcast the results to all other processors on the same column. After all the contributions are collected the final sums are computed at each processor, thus each column of processors have a *replicated* set of Legendre coefficients. The replicated data will be used to increase the parallelism when computing the derivatives for the shallow water equations using finite difference approximations.

Although frequency domain has evenly distributed amount of work, transformation to the Legendre coefficient space does not yield in a same number of coefficients for each of the wavenumbers. To balance the work, the appropriate combinations of wavenumber columns with varying workload are assigned to each group of processors. Note that this has been predetermined from the frequency domain. Thus one mapping from a domain affects the data mapping of another domain significantly, and can be used help to reduce the cost of load balancing or data redistribution.

At the spectral domain, simpler finite different approximation is performed on the dataset, followed by a series of inverse transformations to map the approximated computation back to the physical domain. Application sees the required output represented in physical domain.