Applying Architectural Patterns for Parallel Programming Solving the Two-dimensional Wave Equation

Jorge L. Ortega-Arjona

[1]Departamento de Matemáticas Facultad de Ciencias, UNAM
jloa@fciencias.unam.mx

Abstract. *The Architectural Patterns for Parallel Programming is a collection of patterns related with a method for developing the coordination structure of parallel software systems. These architectural patterns are applied based on (a) the available parallel hardware platform, (b) the parallel programming language of this platform, and (c) the analysis of the problem to solve, in terms of an algorithm and data.*

In this paper, it is presented the application of the architectural patterns along with the method for developing a coordination structure for solving the Two-dimensional Wave Equation. The method used here takes the information from the problem analysis, applies an architectural pattern for the coordination, and provides some elements about its implementation.

This paper is aimed to those who are working with the Patterns for Parallel Software Design. Nevertheless, it presents only a part of the method, at the architectural level, for solving the Two-dimensional Wave Equation. Other two further design issues should be addressed at the communication and synchronization levels, which are not presented here.

1. Introduction

A parallel program is *the specification of a set of processes executing simultaneously, and communicating among themselves in order to achieve a common objective* [18, 19]. This definition is obtained from the original research work in parallel programming provided by E.W. Dijkstra [5], C.A.R. Hoare [9], P. Brinch-Hansen [2], and many others, who have established the main basis for parallel programming today. Practitioners in the area of parallel programming recognize that the success of a parallel program is able to achieve –commonly, in terms of performance– is affected by three main factors: *(a)* the hardware platform, *(b)* the programming language, and *(c)* the problem to solve.

Nevertheless, parallel programming still represents a hard problem to the software designer and programmer: we do not yet know how to solve an arbitrary problem efficiently on a parallel system of arbitrary size. Hence, parallel programming, at its actual stage of development, does not (cannot) offer universal solutions, but tries to provide some simple ways to get started. By sticking with some common parallel *coordinations*, it is possible to avoid a lot of errors and aggravation. Many approaches have been presented up to date, proposing descriptions of top-level coordinations observed in parallel programs. Some of these descriptions are: *Outlines of the Program* [4], *Programming Paradigms* [10], *Parallel Algorithms* [7], *High-level Design Strategies* [11], and *Paradigms for Process Interaction* [1]. These descriptions provide common overall coordinations such as, for example,“master-slave”, “pipeline”, “work-pile”, and others. They represent assemblies of parallel software components which are allowed to simultaneously execute and communicate. Furthermore, these descriptions are expected to support the design of parallel programs, since all of them introduce common forms that such assemblies exhibit.

The *Architectural Patterns for Parallel Programming* [12, 13, 14, 15, 16, 17, 19] represent a Software Patterns approach for designing the coordination of parallel programs. These Architectural Patterns attempt to save the transformation “jump” between algorithm and program. They are defined as *fundamental organizational descriptions of common top-level structures observed in parallel software systems* [12, 18, 19], specifying properties and responsibilities of their sub-systems, and the particular form in which they are assembled together into a coordination.

Architectural patterns allow software designers and developers to understand complex software systems in larger conceptual blocks and their relations, thus reducing the cognitive burden. Furthermore, architectural patterns provide several “forms” in which software components of a parallel software system can be structured or arranged, so the overall coordination of such a software system arises. Architectural patterns also provide a vocabulary that may be used when designing the overall coordination of a parallel software system, to talk about such a structure, and feasible implementation techniques. As such, the Architectural Patterns for Parallel Programming refer to concepts that have formed the basis of previous successful parallel software systems.

The most important step in designing a parallel program is to think carefully about its overall coordination. The Architectural Patterns for Parallel Programming provide descriptions about how to organize a parallel program, having the following advantages [12, 13, 14, 15, 16, 17, 18, 19]:

- The Architectural Patterns for Parallel Programming (as any Software Pattern) provide a description that links a problem statement (in terms of an algorithm and the data to be operated on) with a solution statement (in terms of an organization or coordination of communicating software components).
- The partition of the problem is a key for the success or failure of a parallel program. Hence, the Architectural Patterns for Parallel Programming have been developed and classified based on the kind of partition applied to the algorithm and/or the data present in the problem statement.
- As a consequence of the previous two points, the Architectural Patterns for Parallel Programming can be applied depending on characteristics found in the algorithm and/or data, which drive the selection of a potential parallel structure by observing and studying the characteristics of order and dependence among instructions and/or datum.
- The Architectural Patterns for Parallel Programming introduce parallel structures or coordinations as forms in which software components can be assembled or arranged together, considering the different partitioning ways of the algorithm and/or data.

Nevertheless, even though the Architectural Patterns for Parallel Programming have these advantages, they also present the disadvantage of not describing, representing, or producing a complete parallel program in detail. Other Software Patterns are still needed for achieving this. Anyway, the Architectural Patterns for Parallel Programming are proposed as a way of helping a software designer to apply a parallel structure as a starting point when designing a parallel program.

In summary, the Architectural Patterns for Parallel Programming are briefly described as follows [12, 13, 14, 15, 16, 17, 18, 19]:

1. The *Pipes and Filters pattern* is an extension of the original Pipes and Filters pattern for parallel systems, using a functional parallelism approach where computations follow a precise order of operations on ordered data. Commonly, each component represents a different step of the computation and data is passed from one computation stage to another along the whole structure.
2. The *Parallel Hierarchies pattern* extends the original approach of th e Layers Pattern for parallel systems, considering a functional parallelism in which the order of operations on data is the importan t feature. Parallelism is introduced when two or more hierarchies of layers are able to run simultaneously, performing the same computation.
3. The *Communicating Sequential Elements pattern* is used when the desig n problem at hand can be understood in terms of domain parallelism. The same ope rations are performed simultaneously on different pieces of ordered data. Operations in each component depend on partial results in neighbour components. Usually, this pattern is presented as a network or logical structur e, conceived from the ordered data.
4. The *Manager-Workers pattern* can be considered as a variant of the Ma ster-Slave pattern for parallel systems, introducing an activity p arallelism where the same operations are performed on ordered data. Each compone nt performs the same operations, independent of the processing activity of other components. Different pieces of data are processed simultaneously. Preserving t he order of data is the important feature.
5. The *Shared Resource pattern* is a specialisation variant of the Black board pattern introducing activity parallelism characteristics, wh ere different computations are performed simultaneously, without a prescribed order, on ordered data.

Table 1 classifies the Architectural Patterns for Parallel Programming, regarding their type of parallelism and their homogeneous or heterogeneous nature.

	Functional Parallelism	**Domain Parallelism**	**Activity Parallelism**
Heterogeneous Processing	Pipes and Filters		Shared Resource
Homogeneous Processing	Parallel Hierarchies	Communicating Sequential Elements	Manager-Workers

Table 1: Classification of the Architectural Patterns for Parallel Programming.

For a complete exposition of the Architectural Patterns for Parallel Programming, refer to [12, 18, 19], and further work on each particular architectural pattern in [13, 14, 15, 16, 17].

2. Problem Analysis – The Two-dimensional Wave Equation

The present paper attempts to demonstrate the use of the Architectural Patterns for Parallel Programming for designing a coordination structure that solves the Two-dimensional Wave Equation. The objective is to show how an architectural pattern can be applied so it deals with the functionality and requirements present in this problem.

However, in order to apply any architectural pattern for parallel programming to develop the coordination, several issues about the problem and its constraints should be briefly presented in this section. Hence, the Two-dimensional Wave Equation is described here considering two different points of view: *(a)* from the point of view of the user, which is described in the "Problem Statement" section, and *(b)* from the point of view of the developer, who requires a more computational description of the problem, which is presented in the "Specification of the Problem" section.

2.1. Problem Statement

Partial differential equations are commonly used to describe physical phenomena that continuously change in space and time. One of the most studied and well known of such equations is the Wave Equation, which mathematically models the movement of a region that exposes certain dimensionality, with certain fixed points on its boundaries. In the present example, the region is represented by a two-dimensional entity, for example, a surface of homogeneous material and uniform thickness. The surroundings of the surface are perfectly fixed, and on every extreme, each point keeps a known, fixed position. As the points on the surface moves up and down, their position eventually reaches a value or state in which such a point has a steady, time-independent position maintained by the motion. Thus, the problem of solving the Two-dimensional Wave Equation is to define the equilibrium position of a point $h(i, j)$ for each time-step on the two-dimensional surface. Normally, the wave is studied as a variation of height through an elementary piece of the surface, a finite element. This is represented as a small, two-dimensional element of the surface, with a given size.

Any water surface is an example of the behavior modelled by the Two-dimensional Wave Equation. Particularly at open air, water surfaces produce waves, which depend on the height of the water on a particular coordinate, at a particular time (Figure 1).

In order to develop a program that models the Two-dimensional Wave Equation, first it is necessary to obtain its discrete form. So, the surface is divided into elements, each element with an area of a^2. This area is relatively very small regarding the size of the whole surface, so the element can be considered as a single point within the surface. So, this results on a divided surface, in which three types of elements can be considered (Figure 2).

1. Interior elements, which require computing their position, each one having to satisfy the Two-dimensional Wave Equation.

Figure 1. A water surface, as an example of the Two-dimensional Wave Equation.

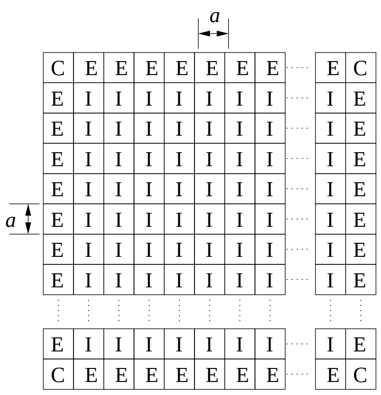

Figure 2. A divided surface with three types of elements: interior (I), exterior (E), and corner (C).

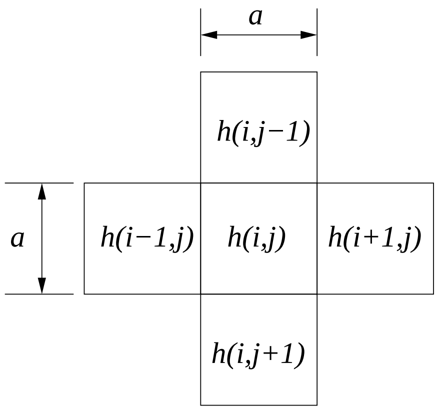

Figure 3. An element $h(i, j)$ and its four neighboring elements.

2. Extreme elements, which are fixed.
3. Corner elements, which are not considered for the simulation.

The discrete solution of the Two-dimensional Wave Equation is based on the idea that the motion of interior elements is due to the height differences between an element and all its neighbors. Let us suppose the position or height of a single interior element $h(i, j)$, whose four adjacent neighboring elements are $h(i-1, j)$, $h(i, j-1)$, $h(i+1, j)$ and $h(i, j+1)$ (Figure 3).

Notice that for the case, a should be small enough so each neighboring element's position can be approximated. So, the discrete two-dimensional Wave Equation is reduced to a difference equation. Rearranging it, it is noticeable that the position of a single element in the next time-step is given as:

$$h(i, j, t+1) \approx h(i, j, t) - h(i, j, t-1) + \frac{1}{4}(h(i+1, j, t) + h(i-1, j, t) + h(i, j+1, t) + h(i, j-1, t))$$

This is the discrete equation to be used in order to obtain a parallel numerical solution for the Two-dimensional Wave Equation.

2.2. Specification of the Problem

From the previous section, it is noticeable that using a surface divided into n^2 elements, the discrete form of the Wave Equation implies a computation for each discrete element of the surface. Moreover, taking into consideration the time as another dimension, so the evolution of temperatures through time can be observed, and solving it using a direct method on a sequential computer, requires something like $O(n^4)$ units of time. Suppose a numerical example: for a surface with, for example, $n^2 = 65536$, it is required to solve about the same number of average operations, involving floating point coefficients. Using a simple sequential computer with a clock frequency of about 1MHz, it would take about eight years for the computation. Furthermore, notice that naive changes to the requirements (which are normally requested when performing this kind of simulations) produce drastic (exponential) increments of the number of operations required, which at the same time affects the time required to calculate this numerical solution.

- *Problem Statement.* The Two-dimensional Wave Equation, in its discrete representation, and for a relatively large number of elements in which a surface is divided, can be computed in a more efficient way by:
  1. using a group of software components that exploit the two-dimensional logical structure of the surface, and
  2. allowing each software component to simultaneously calculate the position value for all elements of the surface at a given time step.

  The objective is to obtain a result in the best possible time-efficient way.

- *Descriptions of the data and the algorithm.* The relatively large number of elements in which the surface is divided and the discrete representation of the Two-dimensional Wave Equation is described in terms of data and an algorithm. The divided region is normally represented as a large surface in terms of a matrix of $(n+2) \times (n+2)$ of elements, which represent every discrete point of the surface, and encapsulate some floating point data which represents its position, as shown as follows. Thus, a whole surface consists of n^2 interior elements and $4n$ exterior elements.

  ```
  class Element implements Runnable{
     ...
     private int i = -1;
     private int j = -1;
     ...
     private Element(int i, int j){
        this.i = i;
        this.j = j;
        new Thread(this).start();
     }
     ...
  }
  ```

  Each **Element** object is able to compute a local discrete wave equation as a single thread. Thus, it exchanges messages with its neighboring elements (whether interior or exterior) and computes its local position, as follows:

  ```
  class Element implements Runnable{
     ...
     private int i = -1;
     private int j = -1;
     ...
     public void run(){
        double [] position, received, total;
        position = new position[3];
        ...
        for (int k = 0; k < iterations; k++) {
           // Here the actual element exchanges data with
           // its neighboring elements
           total = 0.0;
           for (int n = 0; n < 4; n++) {
              // Receive from neighboring elements
              // and put it in the variable ‘received’
  ```

```
                total += received;
            }
            position[k] += position[k-1] - position [k-2] + (1/4 * received)
        }
    }
    ...
}
```

Each time step, a new position for the local **Element** object is obtained from the previous positions and the positions of the neighboring elements (whether interior or exterior). Notice that the term “time step” implies an iterative method in which the operation requires four coefficients. The algorithm described takes into consideration an iterative solution of operations, known as *relaxation*. The simplest relaxation method is the Jacobi relaxation [10, 7, 1], in which the position of each and every interior element is simultaneously approximated using its local previous positions and the positions of its neighbors (and it is the one presented here). Other relaxation methods include the Gauss-Seidel relaxation [1] and the successive over-relaxation (SOR) [1]. Iterative methods tend to be more efficient than direct methods.

- *Information about parallel platform and programming language*. The parallel system available for this example is a SUN SPARC Enterprise T5120 Server. This is a multi-core, shared memory parallel hardware platform, with 1×8-Core UltraSPARC T2, 1.2 GHz processors (capable of running 64 threads), 32 Gbytes RAM, and Solaris 10 as operating system [20]. Applications for this parallel platform can be programmed using the Java programming language [7, 8].
- *Quantified requirements about performance and cost*. This application example has been developed in order to test the parallel system described in the previous point. The idea is to experiment with the platform, testing its functionality in time, and how it maps with a domain parallel application. So, the main objective is simply to test and characterize performance (in terms of execution time) regarding the number of processes/processors involved in solving a fixed size problem. Thus, it is important to retrieve information about the execution time considering several configurations, changing the number of processes on this parallel, shared memory platform.

2.3. A simple example

In order to clarify how the discrete equation presented in Section 2.1 works, so a parallel numerical solution for the Two-dimensional Wave Equation can be obtained, this section presents a simple (but interesting) numerical example of how the discrete equation is used.

For this example, let us consider the simplest discretization of a two-dimensional surface: a 5×5 mesh, which takes into consideration the three types of of elements (as shown in Figure 2), but considering the simplest case (Figure 4):

- 9 interior (I),
- 12 exterior (E), and
- 4 corner (C).

h(0,0)	h(0,1)	h(0,2)	h(0,3)	h(0,4)
h(1,0)	h(1,1)	h(1,2)	h(1,3)	h(1,4)
h(2,0)	h(2,1)	h(2,2)	h(2,3)	h(2,4)
h(3,0)	h(3,1)	h(3,2)	h(3,3)	h(3,4)
h(4,0)	h(4,1)	h(4,2)	h(4,3)	h(4,4)

Figure 4. A simple example of a discretized surface with three types of elements: 9 interior (I), 12 exterior (E), and 4 corner (C).

As mentioned in Section 2.1, the corner elements are not used for the calculation. However, the position of exterior elements are needed to calculate the position of interior elements. So, the position value of exterior element is fixed to zero. Finally, the discrete Two-dimensional Wave Equation is needed to calculate the positions of all the interior elements. Moreover, this example considers a central interior element ($h(2,2)$), whose motion depends directly on its previous two positions as well as the positions of its neighboring interior elements.

Also, in the time dimension, the example requires some initial values of the positions of all the interior elements. For these, some random integer values are used in time-steps 0 and 1. Hence, applying these initial values along with the fixed values of the exterior elements, the discrete Two-dimensional Wave Equation is used to obtain the position values of the interior elements for three more time-steps. Of course, the execution can be extended to a larger number of time-steps, but this simple example only attempts to show what is going on in the simplest terms. The position values for all the interior elements, for 5 time-steps, are shown in the following Table.

Time step	$h(1,1)$	$h(1,2)$	$h(1,3)$	$h(2,1)$	$h(2,2)$	$h(2,3)$	$h(3,1)$	$h(3,2)$	$h(3,3)$
0	3.0	4.0	8.0	6.0	2.0	1.0	0.0	5.0	3.0
1	2.0	3.0	6.0	5.0	3.0	5.0	1.0	6.0	2.0
2	1.0	1.75	0.0	0.5	5.75	6.75	3.75	2.5	1.75
3	-0.4375	0.4375	-3.875	-1.875	5.625	3.625	3.5	-0.6875	2.0625
4	-1.796875	-0.984375	-2.859375	-0.203125	0.25	-2.171875	-1.140625	-0.390625	1.046875

Notice how the position of each interior element (represented as $h(i,j)$, where $i = 1,2,3$ and $j = 1,2,3$) changes through time as the expression of the discrete Two-dimensional Wave Equation dictates. So, the next position of an interior element at a given time-step depends on *(a)* the previous position of such an interior element, *(b)* the actual position of such an interior element, and *(c)* the average of the actual positions of the neighboring elements (whether interior or exterior) of such an interior element. Thus, the positions at time-step 2 are obtained from the positions at time-steps 0 and 1; the positions at time-step 3 are obtained from the positions at time-steps 1 and 2; the positions at time-step 4 are obtained from the positions at time-steps 2 and 3; and so on. This behavior of the discrete Two-dimensional Wave Equation is precisely what we are trying to numerically model here.

3. Coordination Design

In this section, the *Architectural Patterns for Parallel Programming* [12, 18, 19] are used along with the the information from the Problem Analysis, in order to apply an architectural pattern for developing a coordination that solves the Two-dimensional Wave Equation.

3.1. Specification of the System

- **The scope**. This section aims to describe the basic operation of the parallel software system, considering the information presented in the Problem Analysis step about the parallel system and its programming environment. Based on the problem description and algorithmic solution presented in the previous section, the procedure for applying an architectural pattern for a parallel solution to the Two-dimensional Wave Equation problem is presented as follows [12, 19]:
    1. *Analyze the design problem and obtain its specification.* Analyzing the problem description and the algorithmic solution provided, it is noticeable that the calculation of the Two-dimensional Wave Equation is a step-by-step, iterative process. Such a process is based on calculating the next position of each element of the surface through each time step. The calculation uses as input two previous position, and the positions of the four neighbor elements of the surface, and provides the position at the next time step.
    2. *Select the category of parallelism.* Observing the form in which the algorithmic solution partitions the problem, it is clear that the surface is divided into elements, and computations should be executed simultaneously on different elements. Hence, the algorithmic solution description implies the category of **Domain Parallelism**.
    3. *Select the category of the nature of the processing components*. Also, from the algorithmic description of the solution, it is clear that the position of each element of the surface is obtained using exactly the same calculations. Thus, the nature of the processing components of a probable solution for the Two-dimensional Wave Equation, using the algorithm proposed, is certainly a **Homogeneous** one.
    4. *Compare the problem specification with the architectural pattern's Problem section.* An Architectural Pattern that directly copes with the categories of domain parallelism and the homogeneous nature [12, 18, 19] of processing components is the **Communicating Sequential Elements (CSE) pattern** [13, 19]. In order to verify that this architectural pattern actually copes with the Two-dimensional Wave Equation problem, let us

compare the problem description with the Problem section of the CSE pattern. From the CSE pattern description, the problem is defined as [13, 19]:

> *"A parallel computation is required that can be performed as a set of operations on regular data. Results cannot be constrained to a one-way flow among processing stages, but each component executes its operations influenced by data values from its neighboring components. Because of this, components are expected to intermittently exchange data. Communications between components follow fixed and predictable paths".*

Observing the algorithmic solution for the Two-dimensional Wave Equation, it can be defined in terms of calculating the next position of the surface elements as ordered data. Each element is operated almost autonomously. The exchange of data or communication should be between neighboring elements of the surface. So, the CSE is chosen as an adequate solution for the Two-dimensional Wave Equation, and the architectural pattern selection is completed. The design of the parallel software system should continue, based on the Solution section of the CSE pattern.

- **Structure and dynamics**. Based on the information of the Communicating Sequential Elements architectural pattern, it is used here to describe the solution to the Wave Equation in terms of this architectural pattern's structure and behavior.
  1. *Structure*. Using the Communicating Sequential Elements architectural pattern for the Two-dimensional Wave Equation, the same operation is applied simultaneously to obtain the next position values of each element. However, this operation depends on the partial results in its neighboring elements. Hence, the structure of the actual solution involves a regular, two-dimensional, logical structure, conceived from the surface of the original problem. Therefore, the solution is presented as a two-dimensional network of elements that follows the shape of the surface. Identical components simultaneously exist and process during the execution time. An Object Diagram, representing the network of elements that follows the two-dimensional shape of the surface and its division into elements, is shown in Figure 5.

  2. *Dynamics*. A scenario to describe the basic run-time behavior of the Communicating Sequential Elements pattern for solving the Two-dimensional Wave Equation is shown as follows. Notice that all the elements, as basic processing software components, are active at the same time. Every element performs the same position operation, as a piece of a processing network. However, for the two-dimensional case here, each element object communicates with its neighbors as shown in Figure 6.

     The processing and communicating scenario is as follows:
     - Initially, consider only a single **Element** object, **e(i,j)**. At first, it exchanges its local temperature value with its neighbors **e(i-1,j)**, **e(i+1,j)**, **e(i,j-1)**, and **e(i,j+1)** though the adequate communication **Channel** components. After this, **e(i,j)** counts with the different positions from its neighbors.

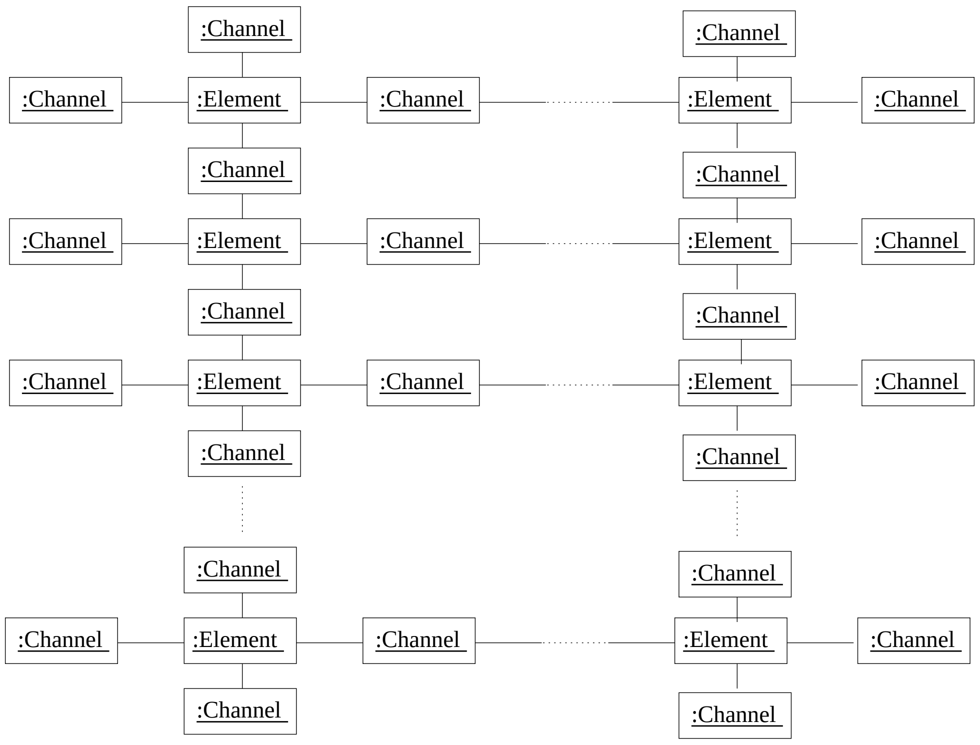

Figure 5. Object Diagram of Communicating Sequential Elements for the solution to the Two-dimensional Wave Equation.

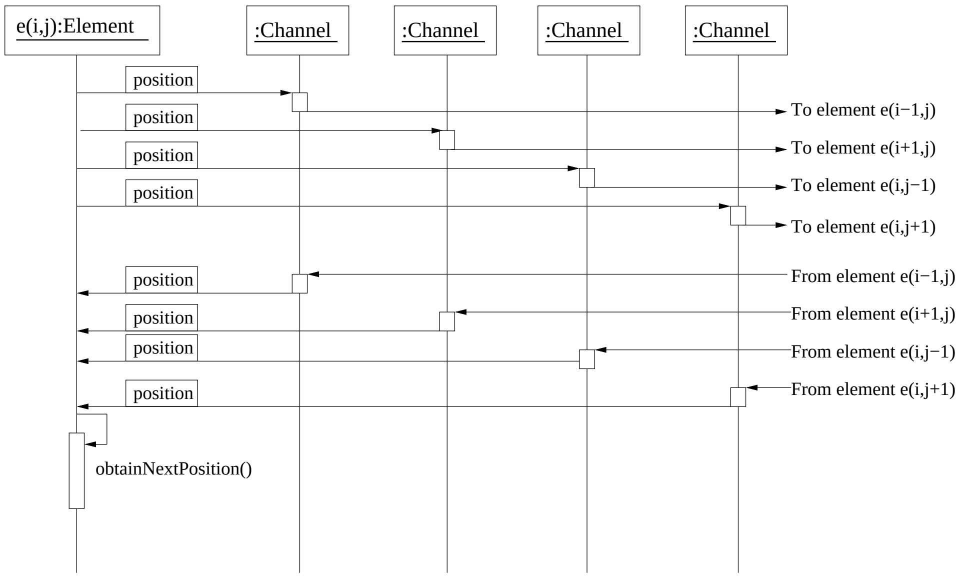

Figure 6. Sequence Diagram of the Communicating Sequential Elements for communicating positions through channel components for the Two-dimensional Wave Equation.

  - The position operation is simultaneously started by the **e(i,j)** component and all the other components of the surface.
  - In order to continue, all components iterate as many times as required, exchanging their partial position values through the available communication channels.
  - The process repeats until each component has finished iterating, and thus, finishing the whole Two-dimensional Wave Equation computation.
3. *Functional description of components*. This section describes each processing and communicating software components as participants of the Communicating Sequential Elements architectural pattern, establishing its responsibilities, input and output for solving the Two-dimensional Wave Equation.
   - **Element**. The responsibilities of an element, as a processing component, are to obtain the next position from the position values it receives, and make available its own position value so its neighboring components are able to proceed.
   - **Channel**. The responsibilities of every channel, as a communication component, are to allow sending and receiving position values, synchronizing the communication activity between neighboring sequential elements. Channel components are developed as the main design objective of a following step, called "Communication Design", which is not addressed in this paper.
4. *Description of the coordination*. The Communicating Sequential Elements pattern describes a coordination in which multiple **Element** objects act as concurrent processing software components, each one applying the same position operation, whereas **Channel** objects act as communication software component which allow exchanging position values between sequential components. No position values are directly shared among **Element** objects, but each one may access only its own private position values. Every **Element** object communicates by sending its position value from its local space to its neighboring **Element** objects, and receiving in exchange their position values. This communication is normally asynchronous, considering the exchange of a single position value, in a one to one fashion. Therefore, the data representing the whole two-dimensional surface represents the regular logical structure in which data of the problem is arranged. The solution, in terms of a divided surface, is presented as a network that actually reflects this logical structure in the most transparent and natural form [13, 19].
5. *Coordination analysis*. The use of the Communicating Sequential Elements patterns as a base for organizing the coordination of a parallel software system for solving the Two-dimensional Wave Equation has the following advantages and disadvantages:
   - **Advantages**
     (a) The order and integrity of position results is granted because each **Element** object accesses only its own local position value, and no other data is directly shared among

components.
(b) All **Element** objects have the same structure and behavior, which normally can be modified or changed without excessive effort.
(c) The solution is easily structured in a transparent and natural form as a two-dimensional array of components, reflecting the logical structure of the two-dimensional surface in the problem.
(d) All **Element** objects perform the same position operation, and thus, granularity is independent of functionality, depending only on the size and number of the elements in which the two-dimensional surface is divided. Changing the granularity is normally easy, by just adjusting the number of **Element** objects in which the surface is divided, thus obtaining a better resolution or precision.
(e) The Communication Sequential Elements pattern can be easily mapped into the shared memory structure of the parallel platform available.

– **Liabilities**
(a) The performance of a parallel application for solving the Two-dimensional Wave Equation based on the Communicating Sequential Elements pattern is heavily impacted by the communication strategy used. For the present example, the threads available in the parallel platform have to take care of a large number of **Element** objects, so each thread has to operate on a subset of the data rather than on a single value. Due to this, dependencies between data, expressed as communication exchanges, could be a cause of a slow down in the program execution.
(b) For this example, load balancing is kept by allowing only a fixed number of **Element** objects per thread, which tends to be larger than the number of threads available. Nevertheless, if data would not be easily divided into same-size subsets, then the computational intensity varies on different processors. Even though every processor is virtually equal to the others, maintaining the synchronization of the parallel application means that any thread that slows down should eventually catch up before the computation can proceed to the next step. This builds up as the computations proceeds, and could impacts strongly on the overall performance.
(c) Using synchronous communications implies a significant amount of effort required to get a minimal increment in performance. On the other hand, if the communications are kept asynchronous, it is more likely that delays would be avoided. This is taken into consideration in the next step,

"Communication Design" (not described here).

4. Implementation

In this section, all the software components described in the Coordination Design step are considered for their implementation using the Java programming language. Once programmed, the whole system is evaluated by executing it on the available hardware platform, measuring and observing its execution through time, and considering some variations regarding the granularity.

Here, it is only presented the implementation of the coordination structure, in which the processing components are introduced, implementing the actual computation that is to be executed in parallel. Further design work is required for developing the channel as communication and synchronization components. Nevertheless, this design and implementation goes beyond the actual purposes of the present paper.

The distinction between coordination and processing components is important, since it means that, with not a great effort, the coordination structure may be modified to deal with other problems whose algorithmic and data descriptions are similar to the Two-dimensional Wave Equation, such as the Two-dimensional Heat Equation [7, 3] or the Poisson Equation [6].

4.1. Coordination

Considering the existence of a class `Channel` for defining the communications between **Element** objects, the Communicating Sequential Elements architectural pattern is used here to implement the main Java class of the parallel software system that solves the Two-dimensional Wave Equation. The class `Element` is presented as follows. This class represents the Communicating Sequential Elements coordination for the Two-dimensional Wave Equation example.

```
class Element implements Runnable{
   private static int M = 65536, N = 65536, iterations = 10;
   private static Channel[][][] element = null;
   private int i = -1;
   private int j = -1;
   public Element(int i, int j){
      this.i = i;
      this.j = j;
      new Thread(this).start();
   }
   public void run(){
      double [] position, received, total;
      position = new position[3];
      for (int x = 0; x < 3; x++) position[x] = random(10*M);

      for (int iter = 0; iter < iterations; iter++) {
        // Send local positions to neighbors
        if (i < M-2 && j > 0 && j < N-1) send(element[i+1][j][0], position);
        if (i > 1   && j > 0 && j < N-1) send(element[i-1][j][1], position);
        if (j < N-2 && i > 0 && i < M-1) send(element[i][j+1][2], position);
        if (j > 1   && i > 0 && i < M-1) send(element[i][j-1][3], position);
        total = 0.0;
        // Receive position from neighbors
        if(i > 0 && j > 0 && i < M-1 && j < N-1){
           for(int x = 0; x < 4; i++){
              received = receive(element[i][j][x]);
              total += received;
           }
        }
        // Insert processing here
      }
   }
   public static void main(String[] args){
      segment = new Channel[M][N}[2];
      for(int m = 0; m < M; m++){
         for(int n = 0; n < N; n++){
            for(int p = 0; p < 4; p++){
               element[m][n][p] = new Channel();
            }
         }
      }
      for(int m = 0; m < M; m++){
         for(int n = 0; n < N; n++){
            new Element(m,n);
         }
      }
      System.exit(0);
   }
}
```

This class only creates two adjacent, two-dimensional arrays of `Channel` components and `Element` components, which represents the coordination structure of the whole parallel software system, developed for executing on the available parallel hardware platform. `Channel` components are used for exchanging position values between neighboring `Element` components, each one first sending its own position value (which is an asynchronous, non-blocking operation), and later retrieving the position values of the four neighboring surface components. Using this data, now it is possible to sequentially process to obtain the new position of the present component. This communication-processing activity repeats as many times as iterations defined.

4.2. Processing components

At this point, all what properly could be considered “parallel design and implementation” has finished: data is initialized (here, randomly, but it can be initialized with particular temperature values) and distributed among a collection of `Element` components. It is now the moment to insert the sequential processing which corresponds to the algorithm and data description found in the Problem Analysis, This is done in the class `Element`, where it is commented `Insert processing here`, by simply adding the following code, and considering the particular declarations for its computation:

```
        position[k] += position[k-1] - position [k-2] + (1/4 * received);
```

The simple, sequential Java code allows that each `Element` component obtains a local position based on the Two-dimensional Wave Equation. Modifying this code implies modifying the processing behavior of the whole parallel software system, so the class `Element` can be used for other parallel applications, as long as they are two-dimensional and execute on a shared memory parallel computer.

5. Summary

The Architectural Patterns for Parallel Programming are applied here along with a method in order to show how to apply an architectural pattern that copes with the requirements of order of data and algorithm present in the Two-dimensional Wave Equation problem. The main objective of this paper is to demonstrate, with a particular example, the detailed design and implementation that may be guided by a selected architectural pattern. Moreover, the application of the Architectural Patterns for Parallel Programming and the method for selecting them is proposed to be used during the Coordination Design and Implementation for other similar problems that involve the calculation of differential equations for a two-dimensional problem, executing on a shared memory parallel platform.

6. Ackowledgements

This work is part of an ongoing research in the Departmento de Matemáticas. Facultad de Ciencias, UNAM, funded by project IN109010-2, PAPIIT-DGAPA-UNAM, 2010.

References

[1] G.R. Andrews *Foundation of Multithreaded, Parallel and Distributed Programming.*, Addison-Wesley Longman, Inc., 2000.

[2] P. Brinch-Hansen *Distributed Processes: A Concurrent Programming Concept.*, Communications of the ACM, Vol.21, No. 11, 1978.

[3] P. Brinch-Hansen *Studies in Computational Science. Parallel Programming Paradigms*, Prentice Hall, 1995.

[4] K.M. Chandy, and S. Taylor *An Introduction to Parallel Programming.* Jones and Bartlett Publishers, Inc., Boston, 1992.

[5] E.W. Dijkstra *Co-operating Sequential Processes*, In Programming Languages (ed. Genuys), pp.43-112, Academic Press, 1968.

[6] J. Gazdag and H.H. Wang *Concurrent computing by sequential staging of tasks*, In IBM Systems Journal, pp.646-660, IBM Co., 1989.

[7] S. Hartley *Concurrent Programming. The Java Programming Language.*, Oxford University Press Inc., 1998.

[8] Herlihy, M., and Shavit, N., *The Art of Multiprocessor Programming.* Morgan Kaufmann Publishers. Elsevier, 2008.

[9] C.A.R. Hoare *Communicating Sequential Processes.* Communications of the ACM, Vol.21, No. 8, August 1978.

[10] S. Kleiman, D. Shah, and B. Smaalders *Programming with Threads*, 3rd ed. SunSoft Press, 1996.

[11] B. Lewis and D.J.. Berg *Multithreade Programming with Java Technology*, Sun Microsystems, Inc., 2000.

[12] J.L. Ortega-Arjona and G.R. Roberts *Architectural Patterns for Parallel Programming*, Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing (EuroPLoP98), Kloster Irsee, Germany, 1998.

[13] J.L. Ortega-Arjona *The Communicating Sequential Elements Pattern. An Architectural Pattern for Domain Parallelism*, Proceedings of the 7th Conference on Pattern Languages of Programming (PLoP2000), Allerton Park, Illinois, USA, 2000.

[14] J.L. Ortega-Arjona *The Shared Resource Pattern. An Activity Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 3rd European Conference on Pattern Languages of Programming and Computing (EuroPLoP98), Kloster Irsee, Germany, 1998.

[15] J.L. Ortega-Arjona *The Manager-Workers Pattern. An Activity Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 9th European Conference on Pattern Languages of Programming and Computing (EuroPLoP2004), Kloster Irsee, Germany, 2004.

[16] J.L. Ortega-Arjona *The Parallel Pipes and Filters Pattern. A Functional Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 10th European Conference on Pattern Languages of Programming and Computing (EuroPLoP2005), Kloster Irsee, Germany, 2005.

[17] J.L. Ortega-Arjona *The Parallel Layers Pattern. A Functional Parallelism Architectural Pattern for Parallel Programming*, Proceedings of the 6th Latin American Conference on Pattern Languages of Programming and Computing (SugarLoafPLoP2007), Porto de Galinhas, Pernambuco, Brasil, 2007.

[18] J.L. Ortega-Arjona *Architectural Patterns for Parallel Programming: Models for Performance Evaluation*, PhD Thesis, Department of Computer Science, University College London, UK, 2007. `http://www.sigsoft.org/phdDissertations/theses/JorgeOrtega.pdf`

[19] J.L. Ortega-Arjona *Patterns for Parallel Software Design*, John Wiley & Sons, 2010.

[20] Sun Microsystems. *Sun SPARC Enterprise T5120 Server.* `http://www.sun.com/servers/coolthreads/t5120/`.