Enhancing Data Parallel Aplications with Task Parallelism

Fernandez J., Guerrero R., Piccoli F., Printista M., Villalobos M. *
Departamento de Informática
Universidad Nacional de San Luis
Ejército de los Andes 950
5700 - San Luis
Argentina
e-mail: {jmfer, rag, mpiccoli, mprinti, mavi}@unsl.edu.ar

Abstract

Most parallel applications contain data parallelism and almost all discussion of its solutions has limited to the simplest and least expressive form: flat data parallelism. Several generalization of the flat data parallel model have been proposed because a large number of those applications need a combination of task and data parallelism to represent their natural computation structure and to achieve good performance in their results. Their aim is to allow the capability of combining the easiness of programming of the data parallel model with the efficiency of the task parallel model.

In this work, we examine how to enhance two basic data parallel computation applications with task parallelism. Applications presented: *N-body Simulation* and *Echo Elimination Process* have been chosen from an unlimited scope of applications where the benefit of the integration of task and data parallelism can be shown.

Keywords: Data parallelism, Task parallelism, Nested data parallelism, Pipeline parallelism, All-pairs computations, Image-Processing, Color images.

*Group supported by the UNSL and ANPCYT (Agencia Nacional para la Promoción de la Ciencia y Tecnología)

1 INTRODUCTION

Data parallelism is one of the more successful efforts to introduce explicit parallelism to high level programming languages. The approach is taken because many useful computations can be framed in terms of a set of independent subcomputations, each one strongly associated with an element of a large data structure. Such computations are inherently parallelizable. Data parallel programming is particularly convenient for two reasons: the first, is its easiness of programming and the second is that it can scale easily to larger problem sizes.

Developments in parallelism have primarily focused on data parallelism [8][23] as a means to portable and efficient parallel programming. Most parallel applications contain data parallelism and almost all discussion of its solutions has limited to the simplest and least expressive form: unstructured data parallelism (flat). However, a large number of such applications need a combination of task and data parallelism to represent the natural computation structure or to enhance performance.

The task parallel model achieves parallelism by using multiple threads of control, each one getting a part of the problem. Task parallelism is based on the concept of a *processor subgroup*, which is a collection of processors executing in data parallel manner. An instance of a program statement is executed by a processors subgroup, which may include all processors executing the program, or only a subset of them. Task paral-

lelism is obtained by dividing the current processors into processors subgroup and performing independent parallel computations on disjoint processors subgroups.

The two application presented here (the N-body Simulation and the Echo Elimination Process) have been chosen from an unlimited scope of applications where the benefit of the integration of task and data parallelism is shown and, since they provide an excellent scenario for benchmarking. We illustrate how to enhance two basic data parallel computation structures with task parallelism.

If a data parallel computation is repeated for a sequence of data sets, it is also possible to divide the processors into subgroups and assign the entire processing of each data set to one of the processors subgroups. This kind of parallelism is normally called *Replication*. It is possible to use *Replication* in combination with pipeling to improve scalability of stream based computations.

Many computational problems encountered in practice have an outer-level of coarse grained parallelism, where the number of task are few, but where each task contains a large amount of work. Each such outer-level task might itself be a parallel task of more fine grained parallelism. These kind of problems invite to the use of multilevel parallelism or *nested* parallelism [5].

This paper is organized as follows. Section 2 presents the two referred problems, the following section describe the data, data & task and, task parallelism solutions for the presented applications. The last section contains the conclusions.

2 STUDY CASES

In this section, a brief introduction about the selected problems and the involved concepts are explained. They will try to show how their nature apply for different parallel solutions.

2.1 The N-Body Problem

The original motivation was to simulate the N-Body Problem. This problem is concerned with determining the effects of the forces between "bodies". The objective is to find the positions and movements of the bodies in space that are subject to gravitational forces from others bodies using Newtonian laws of physics[20][24]:

$$\frac{\delta^2 \vec{x}_1}{\delta t^2} = \sum_{j \neq i}^{N} \vec{a}_{ij} = \sum_{j \neq i} -\frac{G m_j \vec{d}_{ij}}{\mid d_{ij} \mid^3}$$

where

$$\vec{d}_{ij} \equiv \vec{x}_i - \vec{x}_j$$

Direct implementation of this system is a trivial programming exercise. It is simply a double loop which vectorizes very well. At discrete time intervals, the algorithm computes the forces on the bodies and adjust their velocities and positions:

While $time \leq t\_end$ do

- ✔ Accumulate forces by finding the force f(i,j) of particle pair i,j (All-pairs force)
- ✔ Integrate the equations of motion by updating the position (leapfrog)
- ✔ Update time counter

Unfortunately, it has an asymptotic time complexity of $O(N^2)$. When there are large number of bodies, the evaluation of the force function can consume too much time. If we have N bodies, then there will be N evaluations of the force function, each of which will have $N\text{-}1$ terms that involve expensive operations. In ours experiments with a system of 256 bodies, the evaluation of the force function will account for well over 95% of the CPU time used. The integration required for to compute new_position and all the other housekeeping taking up around 5%. For other N-body programs that have to evaluate thousands or even millions of bodies, this method of evaluating the force function is almost completely useless. Because the efficient evaluation of the force function is so critical for most N-body programs, this has been a very active area of research in the last 15

years. Many different approaches [2][3] have been used which reduce the overall time and allow simulation of systems with larger values of N, but its trade-off and profit is not seem to be known.

The basic algorithm is shown in Figure 1. We omit declarations of arrays and variables for conciseness when they are not relevant to the discussion. For a computer simulation, we need to produce several generations until to arrive to *t_end* (maximum time of processing).

```
void N−body(Body *particles)
{
    t=0;   /* time variable */
    while (t < t_end) {
        for (i = 0; i < N ; i++) {
            force(particles , i );
            new_position(particles[i]  , dt);
        }
        t+=dt;
        new_tstep(dt);
    }
}
```

Figure 1: N-Body program

If *force(body, i)* process in Figure 1, is implemented in the obvious fashion, the distance to every body must be found for every body. This particular method of evaluating the force function is known as a Particle-Particle Method [24] because each body is treated as a particle and it is checked against other bodies (particles). The force calculus process is shown in Figure 2.

```
void  force(Body *particles, int i)
{ /* The parameter i determines which body will be trated */
    j=0;
    while (j < N) {
        if (particles[j]  !=particles[i]);
            f(particles[i], particles[j] );
        j++;
    }
}
```

Figure 2: Force Calculus Function

An all-pairs computation [15]performs an operation *f(particle[i],particle[j])* on every possible pair of bodies. This operation transforms *particle[i]* without involving any other elements. We will say that the operation defines an interaction between a pair of elements (only *particle[i]* is modified). In our problem all operations on a particular body take place strictly one at the time. There is no possibility of race conditions when the all-pairs computation is performed in parallel.

The choice of time-step used in *new_position* process is critical to the success of any of the integration schemes. In each generation we use intervals being as short as possible to achieve the most accurate solution. If a constant time-step is used, the particle will travel too fast and the errors can get unacceptably large when two particles come close together. The way to solve this problem is to vary the time step during the calculation. It is to say, we considered a numerical integration scheme that uses variables time-step. In our algorithm the time integrator is a simple leapfrog scheme[17]. Its simplicity makes it an attractive alternative. These schemes automatically cut the time-step down when the particles are near each other, and increase the time step when the particles are far away.

2.2 The Color-Image Echo Restoration Problem

Any image-acquisition process is submitted to problems when it is converting images between physical and electronic media, so there exist many different image-processing techniques and algorithms involved on improve or enhance stored digital images [1][10]. The usefulness of each one depends on the problem in particular.

The most common problem is noise, that is, unwanted variations added to the captured original signal. Noise can sometimes be characterized by its effect on the image. This is the case of images with *ghost* (echo). Echo is a particular kind of noise where the combined signal is the original signal convoluted with itself, that was delayed in the time of arrive and has suffered an attenuation in its intensity as a delay consequence.

The echo problem can be resolved thru an image restoration process that makes possible the extraction of the parameters (the *delay* and the *attenuation*) of the signal producing the noise [7][19] and then separates the combined signals.

The whole process involves to accomplish different extended mathematical stages that can be reduced when the images are treated on the frequency domain by using the Fast Fourier Transform. At last, six stages are settled [13]:

- ✔ to compute the Direct Fast Fourier Transform of the image,
- ✔ to calculate the Cepstrum,
- ✔ to compute de Direct Fast Fourier Transform of the Cepstrum
- ✔ to obtain the delay (τ) and the attenuation (α),
- ✔ to apply the Filter and then,
- ✔ to compute the filtered image Inverse Fast Fourier Transform

Because it involves extensive repetitive calculations on large amount of data and the calculating time usually grows linearly with the data size, a computational approach (**E**cho **E**limination **P**rocess) accepts naturally, many different parallel solutions [18][25][26]. While there exist different good and practical approaches, these works are based on grey-scale images because of its characteristics and representation simplifies its manipulation.

Nevertheless, as a result of technological evolution, full color images have turned significant in a broad range of applications. A color may be characterized by its brightness and chromaticity. Owing the structure of the human eye, all colors are seen as variable combinations of the three so-called primary colors: red, green and blue. The amounts of red, green, and blue needed to form any particular color are called the *Tristimulus Values*.
The purpose of the any color model is to facilitate the specification of color in some general and standard accepted way. The model most commonly used in practice for color monitors and a broad class of color video cameras is RGB. An RGB monitor produces colors by mixing different amounts of red, green and blue light.

In a more tidy world, the use of EEP on color images could be thought as an extention of EEP on gray-scale images and this would be the end of the story. Unfortunately, it doesn't work this way. In a gray-scale EEP there exist a range of errors while it is trying to get back the original photometric content of a pixel. In spite of that it takes advantage from a human-eye skill; the eye only have to respond to the sensation of brightness, so it interprets the corresponding value and makes the proper adjustment.
In color images with echo, the echo is reflected over the three color planes, then at first, they must be treat independently in order to get the restored image. If we want to match a particular color at the original image, we just need to find the values of the three corresponding planes RGB. Then we can mix proportional amounts of our lights to produce the desired color but regarding that it must belong to the range of colors that RGB monitors can reproduce and must match the color that human eye can see.

Finally, there exist two aspects that should be considered in a parallel color EEP: the inherent parallelism on the full color images (the color planes' independence, the echo influence onto the image planes and onto the plane pixels) and the inherent parallelism to the whole EEP [7].

3 DATA and TASK PARALLELISM

When speaking about Data parallelism, it must be understood as the use of multiple functional units that apply the same operation simultaneously to elements of a data set. Instead, task or control parallelism is achieved by applying different operations to different data elements simultaneously. Data parallelism is simple for programming and can scale easily to larger problem sizes. Nevertheless, task parallel model achieves parallelism by using multiple threads of control, each one getting a part of the problem. These two types of parallelism have disadvantages: the first one is only applicable to problems where a large set of data has to be uniformly operated and the second one is more difficult to understand and use, but allows efficient implementations of irregular algorithms [23].

Many parallel applications do not completely fit into data parallel model. Although some applications contain data parallelism, task paral-

lelism is needed to represent the natural computation structure or enhance performance. Combining the programming easiness of the data parallel model with the efficiency of the task parallel model allows to obtain parallel forms that represents the nature of the problem [9].

In the following subsections different solutions to the proposed problems are shown, ones using data parallelism and the others combining the two parallel paradigms.

3.1 Data Parallelism Solutions

This section presents a solution to each one of the problems by using data parallelism. The N-body solution applies data pipelining as a basic computation structure while the color EEP uses just a simple replicated data parallelism.

3.1.1 Data Pipeline N-body algorithm

In this section, we describe the N-body algorithm using the pipeline technique [21]. We will solve the N-body problem simulating a pipeline with *Np* nodes or stages, where 1 $\leqslant$Np$\leqslant$N. The nodes communicate only through data messages. The input data is distributed in $\frac{N}{Np}$ size blocks (*npart*) among the *Np* processors. Each processor establishes communication with other two, it receives data from one of them and sends it to the other in a ring fashion. Figure 3 shows the pipeline:

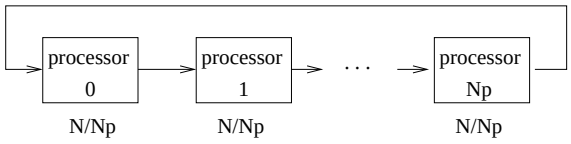

Figure 3: Pipeline N-Body

Figure 4 shows the pseudo-code of the N-body pipeline algorithm:

The algorithm presented here uses a pipeline to compute the forces among all pairs of bodies. First, the pipeline is created and initialized, then the forces are calculated among the local particles. After that, each processor sends data to the proper one and receives data from other. Later, it computes forces with the local and the received

```
void PipeNbody (Body *particles)
{
   Pipe pipe;
   Body *auxp;    /* It will have partial solutions*/
   Body *buffer;  /* It will contain interchanged data of pipe*/
   double dt;       /* time step */

   MPI_Pipe_create(MPI_COMM_WORLD, Np, &pipe);
   npart = N/Np;
   while (t < t_end){
      MPI_Pipe_start(pipe, particles, npart);
      force(particles, auxp, npart, particles, npart);
      buffer=particles;
      for (step=1; step < Np; step++) {
         MPI_Pipe_sendright(pipe, buffer, sizeofbuf);
         MPI_Pipe_receiveleft(pipe, buffer, sizeofbuf);
         force(particles, auxp, npart, buffer, sizeofbuf);
      }
      new_position(*particles, auxp, dt);
      t+=dt;
      new_tstep(dt);
   }
}
```

Figure 4: Pipeline N-body algorithm

data. This task is repeated *Np-1* times, when it finishes a combination phase is necessary. This phase calculates, in each processor and with its own particles, the new particle position in the space. The process is repeated by a fixed period of time, *t_end*.

The *MPI_Pipe* instructions were implemented using standard MPI instructions, such as *MPI_Cart_create*, which creates a new communicator to which topology information has been attached; a ring topology in our case, and *MPI_Cart_shift* which returns the shifted source and destination process ranks. The process ranks which a process receives data from and the one which it sends data to.

3.1.2 Replicated one-plane EEP

If a parallel computation is replicated for different data subsets, it is also possible to consider the whole processing as a unique parallel operation on the data set.

In this case the simplest system arises as a consequence of taking advantage from the RGB model for color images: every plane is saved independently from the others and so they could be treated in that way. The image will be divided in three images (one for each plane), then every one

of these is processed as a gray-scale image and, at the end, the recovered planes are joined into a final image. The Figure 5 shows the referred architecture.

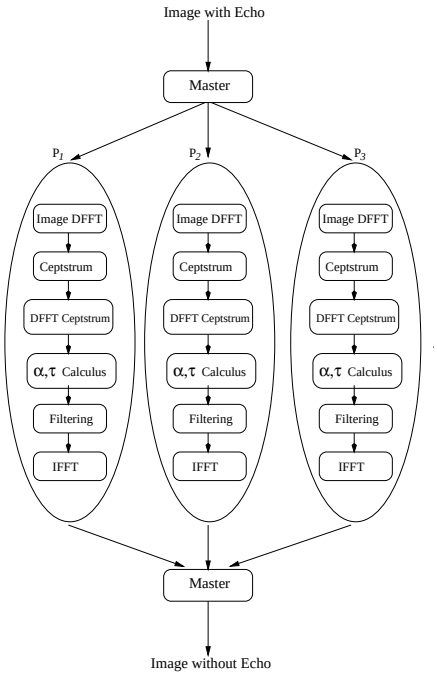

Figure 5: Data Parallelism EEP

Applying EEP to each plane implies an image plane parallel processing where, because their independence, no communication among processes is necessary, so the proposed recovery process is *Embarrassingly Parallel* [26]. At the implementation time, no more than four real processors will be needed: a master and three children.
As the master will be idle during each plane recovering process, this situation can be solved by giving a double identity (master and child) to a particular processor.
The Figure 6 shows the master's pseudocode while the child execute the whole EEP process. The master process has one incoming parameter, the image with echo (*image*), and the outcoming parameter, the image without echo (*IMAGE*).

3.2 Data & Task Parallelism Solutions

In this section, the solutions presented combine data and task parallelism. When considering the N-body problem, the solution is obtained using a Divide and Conquer technique and a nesting

```
void master−echo (Type−image image, Type−image IMAGE)
{
    Type−plane plane[3];              /*each plane of original image */
    Type−plane  PLANE[3];             /* each plane of restored  image*/
     initialize−plane (image, plane)
     for all processor  i of [1..3] do
          PLANE[i]= EEP(plane[i];
     Build−image (PLANE, IMAGE)
}
```

Figure 6: The pseudocode of the master

data parallelism for the computation structure [16]. Instead the Color EEP solution applies a farm style nested data parallelism.

3.2.1 Nested N-body algorithm

We are experimenting with a classic divide-and-conquer algorithm. In this case the input data are replicated in all processors. Before beginning a new generation, the solution to the problem (developed in the previous generation) is required to be also in all processors. The process that accumulates forces by finding the interaction of particle j on particle i, parallelizes easily. The idea is to recursively *divide* the spatially unsorted group of bodies in subgroups until each processor has the responsibility over one group of bodies. In general, this method represents a system of N-bodies in a hierarchical manner. We do not do a spatial domain decomposition. The bodies will be joined in a group in accordance with their initial creation order. This results in a decomposition, which gives each processor an equal amount of work.

Then each processor calculates the total accumulated forces on the bodies belonging to its subgroup and their new position. After that, during *conquer* process, each processor receives the solution from its partner (processor in the same dimension of the binary tree), until that all nodes get the solution of the original problem. Let us consider the program in Figure 1. The internal loop (N) is replaced for a call to *ParNBody(particle,0,N,dt)*. The procedure *ParNBody* is showed in Figure 7. It takes as input a common array of bodies, an initial index, the number of bodies and the time interval. The algorithm pro-

ceeds recursively. The variable *Np* holds the number of processors available at any instant. The algorithm begins testing if there are more than one processor in the current set. If there is only one processor, a call to the sequential algorithm *SeqNBody()*, occurs.

Otherwise, the algorithm is based in partitioning the number of bodies in two subgroups, each one of $npart=\frac{N}{2}$ bodies. For that, the current set of processors is separated in two subsets of processors, each one is going to be the owner of the computation to only one subgroup of bodies. After that, each group makes a recursive call to function *ParNBody()* in order to compute these processes in parallel. Upon the returning, the results are exchanged between partner processors in each group. The *MPI_exchange* instruction was implemented using standard MPI instructions, such as *MPI_sendrecv*.

```
void  ParNbody( Body *particles, int from, int npart, double dt);
{
   int size;
   int from1;
   if (Np >1){
      size = (npart/2)*sizeof(Body);
      from1= from + (npart/2);
      if (name > Np/2) {
         ParNBody(particles, from, npart/2, dt);
         MPI_Barrier(MPI_COMM_WORLD);
         MPI_Exchange(particles+from, size,  partner(name),
                           particles+from1, size,partner(name));
      }
      else {
         ParNBody(particles, from1, npart/2, dt);
         MPI_Barrier(MPI_COMM_WORLD);
         MPI_Exchange(particles+from1, size, partner(name),
                           particles+from, size,  partner(name));
      }
      Np /= 2;
   }
   else
      SeqNbody(particles, from, npart, dt);
}
```

Figure 7: Nested Parallel N-body algorithm

First, *SeqNbody* process has to calculate the force of every particle in its data portion against every other particle in space. Then, it has to adjust the new position (See Figure 8).

```
void SeqNbody(Body *particles, int from, int npart, double dt);
{
   for (i=from; i < from+npart; i++)  /* forces calculus*/
      for(j=0; j < N; j++)
         f(particles[i], particles[j]);
      for (i=from; i < from+npart; i++)  /*new positions calculus*/
         new_position(particles[i],dt);
}
```

Figure 8: Nested Sequential N-body algorithm

3.2.2 Two Level Nested Parallelism in Color EEP

Many computational problems encountered in practice have an outer-level of coarse grained parallelism, where the number of task are few, but where each task contains a large amount of work. Each such outer-level task might itself be a parallel task of more fine grained parallelism. These kind of problems invite to the use of multilevel parallelism or nested parallelism [4][5]. Taking advantage of the intrinsic parallelism on full color images and onto each EEP stage enables to apply nested parallelism; in particular, a two level nested parallelism where each process in the coarse grain (outer level), becomes the supervisor of a group of processors at the inner level. At the first level the processors are divided in three groups or teams (one per plane) of *Np* size, where *Np* has to be power of 2 [9]. Each group is responsible of recovering a particular image plane applying a parallel EEP. Every processor belonging to a group at the inner level will contribute to develop in parallel the corresponding EEP stage [12][13][14]. (See Figure 9).

The Figure 10 shows the pseudocode of the inner level master. The outer level master has the structure showed in the Figure 6, where the EEP operation is replaced by its parallel version.

This approach might be improved considering the independence of the image planes and the fact that the echo reflects with almost the same intensity on each plane (because the used RGB model). Therefore the calculus of the parameters (τ) and (α) should be made just only in one group and the result transmitted to the others.

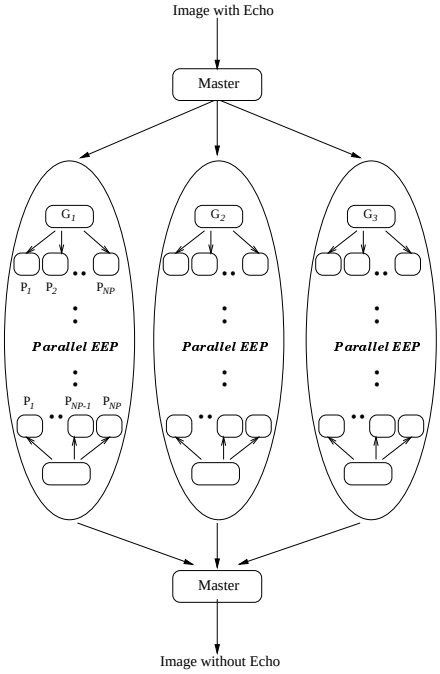

Figure 9: Two Level Nested Parallelism

```
Type plane  innerlevel master (plane)
{
    Parallel−2D−FFT  (plane temporary, FFTDirect−plane)
    Parallel−Cepstrum (FFTDirect−plane, cepstrum−plane)
    Parallel−2D−FFT  (cepstrum−plane, FFT−Cepstrum−plane)
    Calculus the  τ,α;
    Parallel−Filtro (τ,α, FFTDirect−plane, Filtred−plane)
    Parallel−2D−FFT  (Filtred−plane,Final−plane)
    Return (Final−plane)
}
```

Figure 10: The pseudocode of the inner level master

3.3 Task Parallelism Solutions

In this section, the aim is to present solutions that applies task parallelism. In the EEP case, its nature enables the use of this kind of parallelism, in particular, a pure task parallelism. In the N-body case the implemented solution doesn't have a trivial task parallel implementation. Nevertheless, there exist other theoretical solutions (Barnes-Hut algorithm, Fast Multi-pole Method and Anderson's Method) that can be applied [3]. At this moment they have being analyzed, trying to exploit the existence of an inherent task parallelism.

3.3.1 Parallel Pipeline in color EEP

In a *pure* linear pipeline without feedback and forward connections the inputs and outputs are totally independent. In some computation, like linear recurrence, the output of the pipeline are fed back as future inputs. In other words, the inputs may depend on previous outputs. Pipeline with feedback may have a nonlinear data flow. The timing of the feedback inputs becomes crucial to the nonlinear data flow. Improper use of these connections may destroy the inherent advantages of pipelining. On the other hand, proper sequencing with nonlinear data flow may enhance the pipeline efficiency [18].

In this subsection there is an EEP approach as a *general* pipeline with feedback connections. The implementation uses a parallel processing technique that can be viewed as a form of functional decomposition. The EEP is divided into separate functions that must be performed in succession. As it shall see, the input data stream is broken up and processed separately. The stages at the EEP Pipeline execute a specific function (FFT, Ceptrum Calculus, τ and α calculus and Filtering) over the data stream (each plane of the color-image) flowing through the pipe.

Once the pipeline is started up, the processors repeatedly perform computations and data exchanges with other processors. A *Reservation Table* shows how successive pipeline stages are used for the EEP in successive pipeline cycles. (See Figure 11). The pipeline cycle ξ is normalized to the time wasted in a common matrix operation.

	ξ_1	ξ_2	ξ_3	ξ_4	ξ_5	ξ_6	ξ_7	ξ_8	ξ_9	ξ_{10}	ξ_{11}	ξ_{12}
FFT	x	x	x		x	x	x			x	x	x
Cepstrum				x								
Tao & Alfa								x				
Filtering									x			

Figure 11: Reservation Table

It is possible to have multiple marks in a row. Two interesting pipeline-utilization features can be revealed by the reservation table: multiple marks in a row correspond to the repeated usage (marks in far columns) and prolonged usage (for marks in adjacent columns) of the FFT stage.

That table will be used to study various pipeline design problems.
First, multiples X's in a row pose the possibility of collisions. A *Straightforward Greedy Strategy* confirms, that upon the initiation of the first task, the data will have a collision each 1, 2 and 3 pipeline cycles (ξ). The maximum throughput is achieved by an optimal scheduling strategy that achieves minimum average latency without collisions. A good task initiation sequence (without collision) will be each 4 ξ. Unfortunately, that is not good.
Second, the processing speeds of pipeline segments are usually unequal. In the EEP Pipeline, the FFT segment is a *bottleneck*. The throughput of the pipeline is inversely proportional to the bottleneck. Therefore, it is desirable to remove the bottleneck which causes the unnecessary congestion.

When working in parallel with a non-restrictive number of processors, each one of these could implement an specific stage. One design method to increase the throughput is to subdivide the FFT stage in sub-segments. However, if the bottleneck is not sub-divisible or it is not advantageous to divide, it is possible to duplicate the bottleneck in parallel as another way to smooth congestion (See Figure 12).

Nevertheless the control and synchronization of task in parallel segments are much more complex than those for cascaded segments. A more natural way is to repeat three times the FFT stage, transforming the pipeline into a lineal pipeline. It will consist of a cascade of processors or processing stages. Once the pipeline is started up, each processor repeatedly performs computations on a different data set and exchanges data with its neighbor processors.

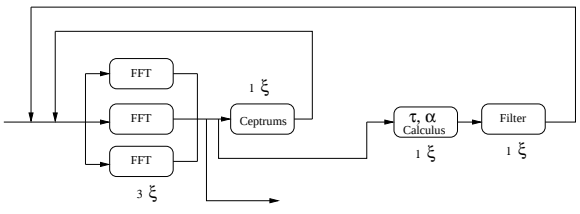

Figure 12: Data Parallel Pipeline

4 CONCLUSIONS

In the last years, integration of task an data parallelism is currently an active area of research and several approaches on different problems have been proposed. Most parallel applications contain data parallelism, but a large number of such applications need a combination of task and data parallelism to represent the natural computation structure or to enhance performance.

In this paper two different and representative problems had been shown and the corresponding solutions proposed by using data parallelism, task parallelism or a combination of them. Image processing applications involve extensive, repetitive and independent mathematical calculus that enables to partition data into small pieces, so a pure data parallel implementation may not scale well. Task parallelism can extend the power of data parallelism and make it significantly more useful. Therefore the combination of task and data parallelism is required to achieve good performance.

The implemented issue for the N-body problem enables data parallelism as an earlier solution while task parallelism best fits when it is needed to recursively divide the available processors to solve the recursively generated subproblems.

Here, we presented different approaches of integrate task and data parallelism into two specific problems. Earlier executions of these suggested solutions show the advantages of integrate them.

Finally, we argue that if a multidisciplinary application is modeled as sets of interacting modules, it can be resolved thru a combination of task and data parallelism.

5 ACKNOWLEDGMENTS

We acknowledge the co-operation of the project group for providing new ideas and constructive criticisms. Also to the Universidad Nacional de San Luis and the ANPCYT from which we receive continuous support.

References

[1] Andrews, H. C. and Hunt B. R. , *Digital Image Restoration*, (Prentice-Hall,1977).

[2] Appel A.W. An efficient program for many body simulation. SIAMJ. COMPUTING. VOL. 6, P. 85, 1985.

[3] Barnes J., Hut P. A hierarchical O(NlogN) force calculation algorithm. nature. Vol 324.P. 446,1986.

[4] Blelloch G. Programming Parallel Algorithms. Communications of ACM. 39(3). March 1996.

[5] Blelloch, G., Hardwick J., Sipelstien j., Zahga M., and Charterjee S. *Implementation of a portable nested data-parallel language.* Journal of Parallel and distributed Computing, 21(1):4-14, April 1994.

[6] Bogert B., Healey M. & Tukey J., "The Frequency Analysis of Time Series for Echoes: Cepstrum, Pseudo-Auto-covariance, Cross-Cepstrum, and Saphe Cracking", *Proc. Symp. Time Series Analysis*, (M. Rosenblatt Ed., J. Wiley & Sons, 1963, pp. 209-243).

[7] Choudhary A. and Ranka S., "Parallel Processing for Computer Vision and Image Understanding", *IEEE Computer*, (Vol 25. Nro. 2, 1992, pp. 7-9).

[8] Foster I., *Designing and Building Parallel Programs*, (Addison Wesley, 1995).

[9] Gonzalez J., León C., Piccoli F., Printista M., Roda J., Rodriguez C. & Sande F., "Groups in Bulk S. Parallel Computing", *Proceedings 8th. Euromicro Parallel and Distributed Processing* (2000, pp. 246-253).

[10] Gonzalez R. and Woods R., *Digital Image Processing*, (Addisson Wesley, 1992).

[11] Guerrero R., Gonzalez A., Zavala E. & Colavita A., "Aplicación de un sistema Homomórfico a la Detección de Ecos en Señales de Video B/N", *Proc. III CACIC, Congreso Argentino de Computación*, (Argentina, 1997, pp 157-164)

[12] Guerrero R., Piccoli F., Printista M., Colavita A., "A Parallel Approach of Image Processing System on PVM" , *Proc. The IASTED International Conference on Computer Systems an Applications*, (Jordan, 1998, pp 17-20)

[13] Guerrero R., Piccoli F., Printista M., "Improvement of a Parallel System for Image Processing" , *Proc. IV CACIC, Congreso Argentino de Computación*, (Argentina, 1998, pp 655-664)

[14] Guerrero R., Piccoli F., Printista M., "Parallelism and Granularity in an Echo Elimination System", *Proc. CSCS-12, 12th International Conference on Control Systems and Computers Science*, IEEE, ISBN 973-96609-5-9, (Romany, 1999, pp 232-237)

[15] Hansen, B. Studies in Computational Science: Parallel Programming Paradigms. Prentice Hall, Englewood, New Jersey. 1995

[16] Hardwick J. An Efficient Implementation of nested data Parallelism for Irregular Divide and Conquer Algorithms. First International Wokshop on High programming Models and Supportive Environments. April 1996.

[17] Hut P., Makino J.,McMIllan S. Building a Better Leapfrog. Institute for Advanced Study, Princeton. NJ 08540. USA.

[18] Hwang K. & Briggs F., *Computer Architecture and Parallel Processing*, (Mc Graw-Hill, 1996).

[19] Lynn P. & Fuerst W., *Digital Signal Processing with Computer Applications*, (J. Wiley & Sons, 1994).

[20] Marciniak, A.Numerical solutions of the N-body Problem. D. Reidel Publishing Co., Dordrecht. 1985.

[21] Module 5. Particle applications Pipeline Computing. http://www.npac.syr.edu/EDUCATION/

[22] Parker J., *Algorithms for Image Processing and Computer Vision*, (J. Wiley & Sons, 1997).

[23] Quinn M. Parallel Computing. Theory and Practice. Second Edition. McGraw-Hill, Inc.

[24] Schlitt, W. The Xstar N-body Solver Theory of Operation. 1996.

[25] Webb J., "Steps Toward Architecture-Independent Image Processing", *IEEE Computer*, (Vol. 25, Nro. 2, 1992, pp. 21-31).

[26] Wilkinson B. & Allen M. - Parallel programming: Techniques and Application using Networked Workstations and Parallel Computers, (Prentice-Hall, 1999).