Parallel Scientific Computing in Distributed Systems: The Paralex Approach

Lorenzo Alvisi **Alessandro Amoroso** **Özalp Babaoğlu**
Alberto Baronio **Renzo Davoli** **Luigi Alberto Giachini**

Technical Report UBLCS-92-2

February 1992

Laboratory for Computer Science
University of Bologna
Piazza di Porta S. Donato, 5
40127 Bologna (Italy)

The University of Bologna Laboratory for Computer Science Research Technical Reports are available via anonymous FTP from the area `ftp.cs.unibo.it:/pub/TR/UBLCS` in compressed PostScript format. Abstracts are available from the same host in the directory `/pub/TR/ABSTRACTS` in plain text format. All local authors can be reached via e-mail at the address *last-name*`@cs.unibo.it`.

UBLCS Technical Report Series

92-1 *Mapping Parallel Computations onto Distributed Systems in Paralex*, by Ö. Babaoğlu, L. Alvisi, A. Amoroso and R. Davoli, January 1992.
92-2 *Parallel Scientific Computing in Distributed Systems: The Paralex Approach*, by L. Alvisi, A. Amoroso, Ö. Babaoğlu, A. Baronio, R. Davoli and L. A. Giachini, February 1992.
92-3 *Run-time Support for Dynamic Load Balancing and Debugging in Paralex*, by Ö. Babaoğlu, L. Alvisi, S. Amoroso, R. Davoli, L. A. Giachini, September 1992.
92-4 *Paralex: An Environment for Parallel Programming in Distributed Systems*, by Ö. Babaoğlu, L. Alvisi, S. Amoroso, R. Davoli, L. A. Giachini, October 1992.
93-1 *Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanism*, by Ö. Babaoğlu and K. Marzullo, January 1993.
93-2 *Understanding Non-Blocking Atomic Commitment*, by Ö. Babaoğlu and S. Toueg, January 1993.

Parallel Scientific Computing in Distributed Systems: The Paralex Approach[1]

Lorenzo Alvisi[2] **Alessandro Amoroso**[3] **Özalp Babaoğlu**[3]
Alberto Baronio[3] **Renzo Davoli**[3] **Luigi Alberto Giachini**[3]

Technical Report UBLCS-92-2

February 1992

Abstract

Modern distributed systems consisting of powerful workstations and high-speed interconnection networks are an economical alternative to special-purpose parallel machines. Unfortunately, current software support requires users to be experts in distributed computing in order to develop parallel applications for such systems. In this paper, we describe the Paralex system that allows parallel scientific applications to be programmed and executed on distributed systems as if they were uniform, multiprocessor parallel machines. We illustrate the main features of the system by considering the Synthetic Aperture Radar signal processing problem. Some preliminary performance results for the system running on a network of Sun-4 workstations are also reported.

1. Appeared as Invited Paper in *Proceedings of the Sixth International Symposium on Computer and Information Sciences*, Side, Antalya, Turkey, October 1991, pp. 1093–1103. Partial support for this work was provided by the Commission of the European Communities under the ESPRIT Programme Basic Research Action Number 3092 (Predictably Dependable Computing Systems) and the Italian Ministry of University, Research and Technology.
2. Department of Computer Science, 4130 Upson Hall, Cornell University, Ithaca, New York 14853 USA.
3. Department of Mathematics, University of Bologna, Piazza Porta S. Donato 5, 40127 Bologna, Italy.

1 Introduction

Economical and technological developments have made workstation-based distributed computing common place. It is not unusual to find sites with dozens, if not hundreds, of general-purpose workstations and a collection of specialized servers. Given the current power of individual workstations and the increasing availability of high-speed local area interconnects such as FDDI and Autonet [17], the differences between multiprocessor parallel computers and distributed systems are diminishing. Thus, it is tempting to try to harness the massive parallelism available in these systems for single, compute-intensive applications. There are, however, several obstacles that remain before networks of workstations can become "a poor man's supercomputer."

Distributed systems differ from multiprocessor parallel computers in that they (i) exhibit no hardware-supported synchrony, (ii) communicate over relatively low-bandwidth, high-latency networks, (iii) lack architectural and linguistic homogeneity, (iv) exhibit increased probability of communication and processor failures, and (v) fall under multiple administrative domains. As a consequence, developing parallel programs in such systems requires expertise in fault-tolerant distributed computing. We claim that current software technology for parallel programming in distributed systems is comparable to assembly language programming for traditional sequential systems—the user must resort to low-level primitives to accomplish data encoding/decoding, communication, remote execution, synchronization, failure detection and recovery. It is clear that effective use of distributed systems as parallel machines requires automated support for handling these complexities.

The Paralex system being developed at the University of Bologna, Italy is an instance of such a system. Paralex can be viewed as a "compiler" that takes graphical descriptions of parallel computations and automatically produces fault-tolerant distributed programs to carry them out. The system is a complete programming environment in that it includes a graphical editor, compiler, run-time support and debugger. The intended application domain is scientific computing since it is best suited to exploit the potential parallelism of distributed systems.

The idea of viewing a collection of workstations on a network as a parallel multiprocessor is a popular one. There are numerous other projects that have been experimenting with different abstractions to provide on top of such a system. For the most part, systems such as Linda [15], Amber [12], `nigen++` [2], AERO [3] and PVM [6] exist as library functions callable by applications and cannot be considered programming environments. On the other hand, systems such as apE [13], CODE [10], FrameWorks [18] and HeNCE [7] include a graphical notation for parallel computations but no support for fault tolerance.

In this paper, we illustrate the utility of the Paralex approach by considering a signal processing application of significant practical interest. The next section gives an overview of Paralex. Section 3 describes the Synthetic Aperture Radar (SAR) signal processing problem and its abstract mathematical formulation. Section 3.2 considers programming SAR as a Paralex application. Performance results obtained by executing the program on a network of Sun-4 workstations is reported in Section 4. Section 5 concludes the paper.

2 Overview of Paralex

Paralex consists of four logical components: A graphics editor for program specification, a compiler, an executor and a runtime support environment. The first three components are integrated within a uniform graphical programming environment. It is, however, possible to run the compiler and the executor manually so that existing Paralex programs can be compiled and executed from machines with no graphics support.

The programming paradigm supported by Paralex is a restricted form of data flow [1]. A Paralex program is composed of *nodes* and *links*. Nodes correspond to computations and the links indicate the flow of (typed) data. Thus, Paralex programs can be thought of as directed graphs (and indeed are visualized as such on the screen) representing the data flow relations plus a collection of ordinary sequential code fragments to indicate the computations. The current prototype limits the structure of the data flow graph to be acyclic.

The semantics associated with this graphical syntax obeys the so-called "strict enabling rule" of data-driven computations in the sense that when all of the links incident at a node contain values, the computation associated with the node starts execution transforming the input data to an output. The computation to be performed by the node must satisfy the "functional" paradigm—multiple inputs, only one output with no side effects. The actual specification of the computation may be done using whatever appropriate notation is available including standard sequential programming languages, parallel programming notations (if the distributed system includes nodes that are themselves multiprocessors), executable binary code or library functions for the relevant architectures.

Unlike classical data flow, the nodes of a Paralex program carry out significant computations. This so-called *coarse-grain* data flow model [5] is a consequence of the underlying distributed system architecture where we seek to keep the communication overhead via a high-latency, low-bandwidth network to reasonable levels.

There are many situations where the single output value produced by a node needs to be communicated to multiple destinations as input so as to create parallel computation structures. In Paralex, this is accomplished simply by drawing multiple output links originating from a node towards the various destinations. To economize on network bandwidth, Paralex introduces the notion of *filter* nodes that allow data values to be extracted on a per-destination basis before they are transmitted to the next node. Conceptually, filters are defined and manipulated just as regular nodes and their "computations" are specified through sequential programs. In practice, however, all of the data filtering computations are executed in the context of the single process that produced the data rather than as separate processes to minimize the system overhead.

Once the user has fully specified the Paralex program by drawing the data flow graph and supplying the computations to be carried out by the nodes, the program can be compiled. The first pass of the Paralex compiler is actually a precompiler to generate all of the necessary stubs to wrap around the node computations to achieve data representation independence, remote communication and replica management for those nodes with fault tolerance needs. Type checking across links is also performed in this phase. Currently, Paralex generates all of the stub code as ordinary C. As the next step, the C compiler is invoked to turn each node into an executable module.

The compiler must also address the two aspects of heterogeneity: Data representation and instruction sets. Paralex uses the ISIS toolkit [9,8] as the infrastructure to realize a universal data representation. All data that is passed from one node to another during the computation are encapsulated as ISIS messages. Heterogeneity with respect to instruction sets is handled in a brute-force manner by invoking remote compilations on the machines of interest and storing multiple executables for the nodes.

As part of the program definition, Paralex permits the user to specify a fault tolerance level for each node of the computation graph. Paralex will generate all of the necessary code such that when a node with fault tolerance k is executed, it will be executed on $k+1$ distinct processors such that the computation will succeed despite up to k failures. The failures that are tolerated are of the benign type for processors (i.e., all processes running on the processor simply halt) and communication components (i.e., messages may be lost). There is no attempt to guard against more malicious processor failures nor against failures of non-replicated components such as an Ethernet cable.

Paralex uses ISIS also as the infrastructure to support fault tolerance. For each node that requires fault tolerance, Paralex creates a process group consisting of replicas for the computation. The type of process group formed is the ISIS *coordinator-cohort* where only one of the replicas (the coordinator) carries out the computation and the others (the cohorts) are invoked automatically only in case of failure of the coordinator. Simple data transmissions from one node to another turn into ISIS reliable broadcast communication from a coordinator to a process group representing the destination node.

Details of how Paralex computation graphs are mapped onto the hosts of a distributed system and dynamic control of the remote execution can be found in [4].

3 The Synthetic Aperture Radar Problem

An important problem in remote sensing is the reconstruction of surface contours despite visual obstructions. One such problem is Synthetic Aperture Radar (SAR) [16] where an aircraft or spacecraft flying at a constant altitude bounces radar signals along the strip of terrain below and records the reflections. Inferring the contours of the terrain despite cloud cover, speed fluctuations, etc. requires complex processing of the resulting data. Reconstructing these images in real-time can only be accomplished using expensive, dedicated signal processing hardware [14].

We have picked the SAR application as a test of effective parallel scientific computing in distributed systems. While it is dangerous to make generalizations based on a single application, we feel that SAR represents a large application domain. In addition, SAR is an instance of a "worst-case" application for distributed systems in the sense that the computation performed in relation to the amount of data distributed is relatively low. Thus, the effectiveness of our approach will have to be judged more by the ease of program development and less by the gains in performance.

3.1 SAR as a Mathematical Problem

The mathematical problem that needs to be solved in order to produce high-quality images from SAR data can be characterized as time domain binary convolution. The input to the problem is a matrix of complex numbers corresponding to the reflection signals from the scanned strip. Informally, the processing can than be summarized as:

Fourier Transform Compute the two-dimensional discrete Fourier transform of the radar signals,

Convolution Compute a point-by-point product of the transformed data and a filter matrix,

Inverse Fourier Transform Compute the two-dimensional inverse discrete Fourier transform of the matrix obtained from the product,

Normalization Compute intensity levels for the image to be visualized by normalizing the resulting data.

To make the above informal description of the algorithm more precise, consider the following definitions. Let N be an integer and let $\vec{x} = (x_0, x_1, \cdots, x_{N-1})$ be a vector of complex numbers. The discrete Fourier transform of vector $\vec{x}$, is a vector $\vec{X} = (X_0, X_1, \cdots, X_{N-1})$, whose components are defined to be:

$$X_k = \sum_{n=0}^{N-1} x_n exp(2\pi ikn/N) \quad , \quad k = 0, 1, \ldots, N-1.$$

Given an integer N and vector $\vec{X} = (X_0, X_1, \cdots, X_{N-1})$ of complex numbers, the inverse discrete Fourier transform of $\vec{X}$, is a vector $\vec{x} = (x_0, x_1, \cdots, x_{N-1})$, whose components are defined to be:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k exp(-2\pi ikn/N) \quad , \quad n = 0, 1, \ldots, N-1.$$

Let $\mathcal{F}(\vec{x})$ denote the discrete Fourier transform of vector $\vec{x}$ and $\mathcal{F}^{-1}(\vec{X})$ its inverse.

The above definitions extend to two dimensions as follows. Let N_1 and N_2 be integers and let $A = (a_{ij})$ be a complex matrix of order $N_1 \times N_2$. The two-dimensional discrete Fourier transform of A is a complex matrix B of order $N_1 \times N_2$ whose elements are defined to be:

$$b_{k,\ell} = \sum_{m=0}^{N_1-1} \sum_{n=0}^{N_2-1} a_{m,n} exp(2\pi ikm/N_1) exp(2\pi i\ell n/N_2)$$

$$k = 0, 1, \ldots, N_1 - 1 \text{ and } \ell = 0, 1, \ldots, N_2 - 1.$$

Similarly, the two-dimensional inverse discrete Fourier transform of a complex matrix B of order $N_1 \times N_2$ is a complex matrix A of order $N_1 \times N_2$ whose elements are defined to be:

$$a_{n,m} = \frac{1}{N_1 N_2} \sum_{k=0}^{N_1-1} \sum_{\ell=0}^{N_2-1} b_{k,\ell} exp(-2\pi ink/N_1) exp(-2\pi im\ell/N_2)$$

$$n = 0, 1, \ldots, N_1 - 1 \text{ and } m = 0, 1, \ldots, N_2 - 1.$$

By observing the definition of the two-dimensional transform, we note that it can be realized as a series of one-dimensional transforms. We first compute N_1 one-dimensional transforms of order N_2 corresponding to the rows of matrix A and then N_2 one-dimensional transforms of order N_1 corresponding to the columns of the resulting matrix. Clearly, one can also switch the order and transform first by column and then by row. Let $A_{k,*}$ and $A_{*,\ell}$ denote the kth row and ℓth column of matrix A, respectively. With slight abuse of notation, we can express the two-dimensional discrete Fourier transform of matrix A as

$$\mathcal{F}((\mathcal{F}(A_{k,*}))_{*,\ell}) \quad , \quad k = 0, 1, \ldots, N_1 - 1 \text{ and } \ell = 0, 1, \ldots, N_2 - 1.$$

The same observation can be made about the inverse two-dimensional transform resulting in the following expression

$$\mathcal{F}^{-1}((\mathcal{F}^{-1}(B_{m,*}))_{*,n}) \ , \ \ m = 0, 1, \ldots, N_1 - 1 \text{ and } n = 0, 1, \ldots, N_2 - 1.$$

Filtering of the radar signal is accomplished by the binary convolution operation in the time domain. Given two complex matrixes $A = (a_{m,n})$ and $B = (b_{m,n})$ both of order $N_1 \times N_2$, the binary convolution of A and B, written $A \otimes B$, results in a complex matrix C whose elements are

$$c_{m,n} = a_{m,n} \cdot b_{m,n} \ , \ \ m = 0, 1, \ldots, N_1 - 1 \text{ and } n = 0, 1, \ldots, N_2 - 1.$$

Finally, we define the normalization operation. Given a complex number c, let $\Re c$ and $\Im c$ denote its real and imaginary components, respectively. The normalization of a complex matrix C of order $N_1 \times N_2$, written $\mathcal{N}(C)$, is a real matrix $R = r_{k,\ell}$ such that

$$r_{k,\ell} = \frac{\sqrt{(\Re c_{k,\ell})^2 + (\Im c_{k,\ell})^2} - min}{max - min}$$

$$k = 0, 1, \ldots, N_1 - 1 \text{ and } \ell = 0, 1, \ldots, N_2 - 1$$

where min and max are the minimum and maximum values, respectively, of the norm $\sqrt{(\Re c_{m,n})^2 + (\Im c_{m,n})^2}$ over all elements of C. The operation is defined only for matrixes where $min \neq max$, which will be the case for non trivial images.

The signal processing computation of SAR can be summarized as follows. Given a radar signal matrix $Radar$ and filter matrix $Filter$, both of order $N_1 \times N_2$, visualize the image corresponding to the pixel intensities of the matrix

$$\mathcal{N}(\mathcal{F}^{-1}((\mathcal{F}^{-1}(\mathcal{F}((\mathcal{F}(Radar_{k,*}))_{*,\ell} \otimes Filter)_{k,*})_{*,\ell})))$$

3.2 SAR Implementation in Paralex

The SAR computation described in the previous section can be easily implemented in Paralex since it fits naturally into the dataflow or functional programming paradigm. Furthermore, each of the computational stages can be easily parallelized up to the smaller dimension of the input matrix by simply partitioning the input data. The only exception to this is the final normalization step that requires a global minimum and maximum to be computed over the entire matrix.

We have implemented the SAR algorithm in Paralex using a standard sequential realization of the Fast Fourier Transform (FFT) [11] code as the basic building block. The resulting computation graph for the 8-way parallel version is shown in Figure 1. For our experiments, the input radar signal matrix is an $N \times N$ square where N is a power of two. The problem can be parallelized to any other dimension that is a power of two up to N by simply modifying the width of the graph. Note that, for a given N, there will be a degree of parallelism beyond which the data communication costs will far outweigh the advantages in computation speedup.

We now briefly describe the computation performed by each stage of the graph in Figure 1 and explain the data communication patterns.

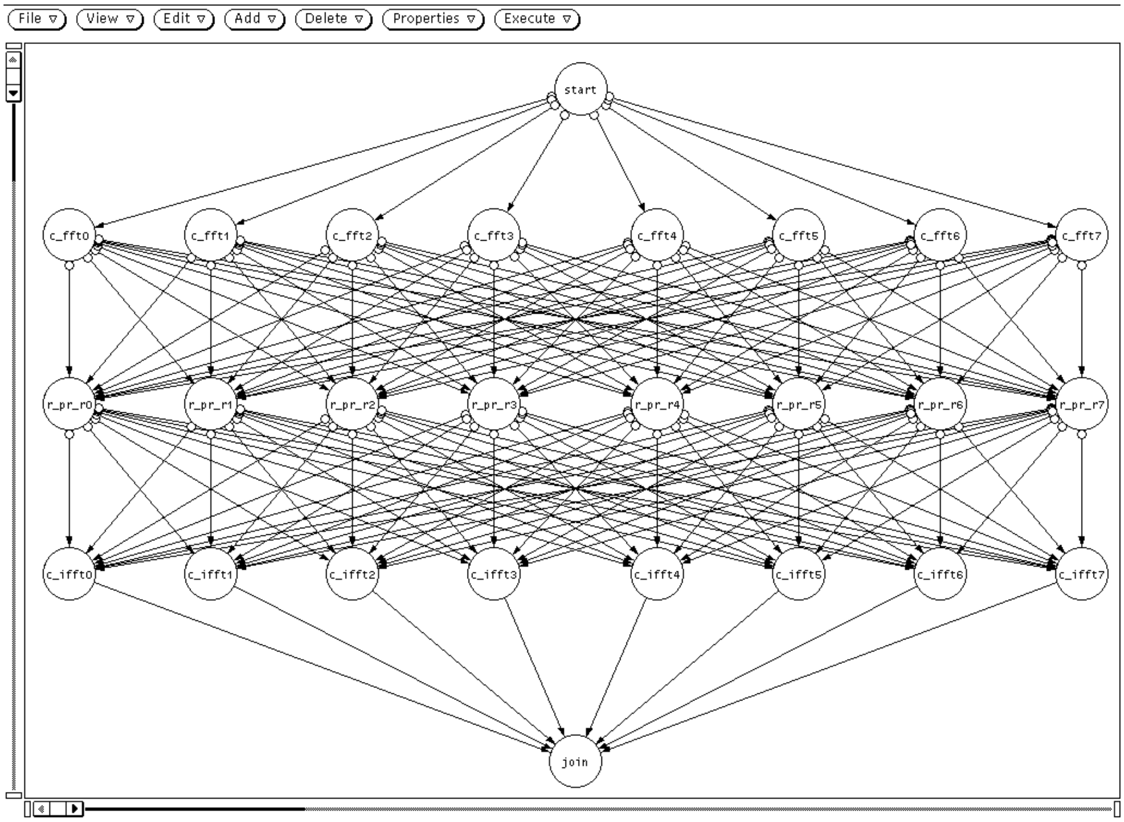

Figure 1. Paralex Computation Graph for the SAR Algorithm.

start The node labeled *start* reads in the $N \times N$ matrix corresponding to the input radar signal. It then broadcasts the matrix to the next level in the computation. As part of the broadcast, each filter (indicated as a small bubble attached to the computation node) extracts an $N \times \frac{N}{8}$ strip of the original matrix. Thus, the data is partitioned among the 8 concurrent nodes of the next level.

c_fft$\{$**0–7**$\}$ Each of these nodes computes $\frac{N}{8}$ FFT's of order N corresponding to the columns of the submatrix received. The result is broadcast to the next stage. The filters again extract elements such that each destination receives only an $\frac{N}{8} \times \frac{N}{8}$ submatrix.

r_pr_r$\{$**0–7**$\}$ The 8 incoming $\frac{N}{8} \times \frac{N}{8}$ submatrixes are composed to form $\frac{N}{8} \times N$ strips on which FFT's are performed. Each node also reads in an $\frac{N}{8} \times N$ strip of the filter matrix and performs the binary convolution. Finally, $\frac{N}{8}$ inverse transforms are computed corresponding to the rows of the resulting matrix. The result is broadcast to the next level. The filters extract elements to form $\frac{N}{8} \times \frac{N}{8}$ partitions.

c_ifft$\{$**0–7**$\}$ The 8 incoming $\frac{N}{8} \times \frac{N}{8}$ submatrixes are composed to form $N \times \frac{N}{8}$ strips on which inverse transforms are performed corresponding to the columns of the resulting matrix. Each element $a_{k,\ell}$ of the submatrix is replaced by its norm given by

Parallelism	*Input Dimension* 64×64	128×128	256×256	512×512
Sequential	1.33	5.64	24.05	104.85
4-way	1.65	3.98	16.50	54.58
8-way	1.55	2.88	8.89	34.40
8-way, pipeline	1.26	3.20	8.49	13.42

Table 1. SAR Execution Times in Seconds

$\sqrt{(\Re a_{k,\ell})^2 + (\Im a_{k,\ell})^2}$. The resulting submatrix along with the local minimum and maximum values are sent to the final level.

join The node computes the final normalized matrix by computing the global minimum and maximum from the 8 local values and performing the necessary arithmetic. The resulting matrix of reals is written to a file.

4 Performance Results

The Paralex program described in Section 3.2 was compiled and run on a network of Sun-4 (SparcStation) workstations. Experiments were performed to compare the performance of the 4 and 8-way parallel implementations of SAR with that of a sequential version. Input data sizes ranged from 64×64 to 512×512 matrixes. Note that in the case of a 512×512 image, the input data (radar signal and filter matrixes) amount to more than 4 Megabytes (8 bytes per complex number). This represents a stress case for most of the resources found in the system—memory, communication bandwidth—as well as the support layer consisting of ISIS.

To map the computation onto the workstations, the nodes of the graph in Figure 1 were grouped into *chains* such that chain i consisted of nodes $\{c\_fft_i, r\_pr\_r_i, c\_ifft_i\}$. Nodes $start$ and $join$ were added to chain 0. Since a chain represents strict sequential data dependency, each chain was mapped to a single processor. Every node of the computation was run as a separate Unix process, each with the ISIS libraries loaded as runtime support. All experiments were performed without any fault tolerance so there were no replicas to map.

Table 1 summarizes the results. Reported numbers are elapsed real times in seconds. The case *Sequential* corresponds to an implementation of SAR as a single process (with no ISIS support for distribution or fault tolerance) and run on a single SparcStation 1. In the case of the parallel versions, chain 0 was mapped to a SparcStation 2 while all other chains were mapped to SparcStation 1′s. The reason for this being that chain 0, which contains 5 nodes, resulted in thrashing due to lack of memory on the SparcStation 1′s with only 8 Megabytes of memory. The SparcStation 2, having 16 Megabytes of memory, was better suited.

While this lack of homogeneity makes the parallel and sequential results more difficult to compare, the results are still meaningful. Given the strict enabling semantics of the dataflow graphs of Paralex, a computation cannot be any faster than the slowest sequential path. Consequently, the fact that nodes $\{c\_fft_0, r\_pr\_r_0, c\_ifft_0\}$ run on a faster machine cannot affect the overall execution time since the final node (*join*) will have to wait for results of the other parallel chains (running on slower machines). The only source of incompatibility of the results is due to nodes *start* and *join* being executed faster. Note, however, that these nodes perform minimal computations.

The results confirm our expectations—the larger the data size, the more significant the speedup. In the case of small input sizes, it hardly pays to parallelize. (In fact, for the 64×64 case, the parallel versions perform worse than sequential.) The overhead of communication and process management outweigh the gain in processing (which is minimal for these dimensions). The final row of the table represents execution times that are averages over 10 consecutive executions of the computation graph. In these pipelined executions it is possible to get parallelism not only across the width of the computation graph but also along the vertical dimension since processors could be working in parallel on different instances of input data. The observed speedup for large data sizes indeed confirms this expectation.

5 Conclusions

The main conclusions to be drawn from the experiment reported are as follows: (i) Distributed systems can form the basis for effective parallel computing, (ii) Much of the complexity associated with parallel computing in distributed systems can be hidden from the user through appropriate support software, (iii) Graphical languages and user interfaces provide a natural way to express parallel computations arising from scientific applications.

Developing the parallel SAR application in Paralex supports these conclusions. The parallel program was produced in a very short amount of time without having to write large amounts of code. The heart of the computation (FFT) was obtained from a mathematical library and simply reused in the various stages. The new code that had to be developed completed the rest of the computation (data extraction, composition, binary convolution and normalization). Furthermore, all of the code that had to be developed by the user was ordinary, sequential code that accomplished well-defined data transformations. Thus, debugging the parallel program could be accomplished by using traditional sequential debugging techniques on the individual pieces of the computation. All of the hard work of producing network-level code was performed automatically by Paralex itself.

Any system that automatically supports higher levels of abstraction must incur a cost. In the case of Paralex, the price to pay for automatic support of data encoding/decoding, communication, remote execution and fault tolerance is performance. The situation is analogous to programming in assembly language to save time and space. It is widely agreed that the advantages of high-level language programming far outweigh the performace penalties. In our case, the current penalties are greater than they ought to be since we incur all of the costs of ISIS for the limited use we make of it. Future versions of Paralex may do away with ISIS entirely and implement light-weight mechanisms needed to support distribution and fault tolerance internally.

References

[1] W. B. Ackerman. Data Flow Languages. *IEEE Computer*, February 1982, pp. 15–22.

[2] R. Anand, D. Lea and D. W. Forslund. Using `nigen++`. Technical Report, School of Computer and Information Science, Syracuse University, January 1991.

[3] D. P. Anderson. The AERO Programmer's Manual. Technical Report, CS Division, EECS Department, University of California, Berkeley, October 1990.

[4] Ö. Babaoğlu, L. Alvisi, A. Amoroso and R. Davoli. Mapping Parallel Computations onto Distributed Systems in Paralex. In *Proc. IEEE CompEuro '91*, Bologna, Italy, May 1991.

[5] R. G. Babb. Parallel Processing with Large-Grain Data Flow Techniques. *IEEE Computer,* July 1984, pp. 55–61.

[6] A. Beguelin, J. J. Dongarra , G. A. Geist, R. Manchek and V. S. Sunderam. A Users' Guide to PVM Parallel Virtual Machine. Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July, 1991.

[7] A. Beguelin, J. J. Dongarra , G. A. Geist, R. Manchek and V. S. Sunderam. Graphical Development Tools for Network-Based Concurrent Supercomputing. In *Proc. Supercomputing '91*, Albuquerque, New Mexico, November 18-22, 1991.

[8] K. Birman and K. Marzullo. ISIS and the META Project. *Sun Technology,* vol. 2, no. 3 (Summer 1989), pp. 90–104.

[9] K. Birman, R. Cooper, T. Joseph, K. Marzullo, M. Makpangou, K. Kane, F. Schmuck and M. Wood. The ISIS System Manual, Version 2.1. Department of Computer Science, Cornell University, Ithaca, New York, September 1990.

[10] J. C. Browne, M. Azam and S. Sobek. CODE: A Unified Approach to Parallel Programming. *IEEE Software,* July 1989, pp. 10–18.

[11] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, vol. 19, 1965.

[12] J. S. Chase, F. G. Amador, E. D. Lazowska, H. M. Levy and R. J. Littlefield. The Amber System: Parallel Programming on a Network of Multiprocessors. In *Proc. ACM Symposium on Operating Systems Principles*, Litchfield Park, Arizona, December 1989, pp. 147–158.

[13] D. S. Dyer. A Dataflow Toolkit for Visualization. *IEEE Computer Graphics and Applications*, July 1990, pp. 60–69.

[14] G. Franceschetti, A. Mazzeo, N. Mazzocca, V. Pascazio and G. Schirinzi. An Efficient SAR Parallel Processor. *IEEE Transaction on Aerospace and Electronic Systems*, Vol. 27, No. 2, March 1991, pp. 343–353.

[15] D. Gelernter, N. Carriero, S. Chandran and S. Chang. Parallel Programming in Linda. In *Proc. Int. Conf. Parallel Processing*, St. Charles, Illinois, August 1985, pp. 255–263.

[16] D. Massonnet and F. Adragna. Synthetic Aperture Radar: New Processing Concepts. In *. Proc. 10th Annual International Geoscience and Remote Sensing Symposium*, Washington, D.C., May 1990, pp. 1323–1326.

[17] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite and C. P. Thacker. Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links. Technical Report 59, DEC Systems Research Center, Palo Alto, California, April 1990. To appear in *IEEE Journal on Selected Areas of Communications*, October 1991.

[18] A. Singh, J. Schaeffer and M. Green. A Template-Based Approach to the Generation of Distributed Applications Using a Network of Workstations. *IEEE Trans. Parallel and Distributed Systems*, vol. 2, no. 1, January 1991, pp. 52–67.