# NESTED PARALLELISM IN PARALLELS PARADIGMS

**Piccoli F., Printista M.**[1]

[1] Universidad Nacional de San Luis
Ejercito de los Andes 950- San Luis
{mpiccoli,mprinti}@unsl.edu.ar
ARGENTINA

**Gonzalez J.A., Leon C., Roda J.L., Rodriguez C. and Sande F.**[2]

[2] Departamento de E.I.O.C.
Universidad de La Laguna
Facultad de Matemáticas. Tenerife
SPAIN

## ABSTRACT

Data parallelism is one of the more successful efforts to introduce explicit parallelism to high level programming languages. This approach is taken because many useful computations can be framed in term of a set of independent sub-computation, each 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 ease of programming and the second is that it can scale easily to larger problem sizes. Several data parallel language implementations now exists [8][9]. However, almost all discussion of data parallelism were limited to the simplest and least expressive form: unstructured data parallelism (flat).
Many other generalizations of the data parallel model have been proposed, which permit the nesting of data parallel constructors to specify parallel computation across nested and irregular data structures[2][3]. These language implementations include the capability of nested parallel invocations, combining the facility of programming on a *data parallel model* with the efficiency in the execution on irregular data structures of the *task parallel model* [1].

The divide and conquer approach is characterized by dividing the problems into subproblems that are of the same structure as the larger problem. Further divisions into still smaller subproblems are usually done by recursion. The recursive method will continually divide a problem until the problem cannot be broken down into smaller parts. Then the very simple tasks are performed. The tasks' results are combined with the others tasks' results in the same level. Nested parallelism is critical for describing divide and conquer algorithms[4][6][10]. A simple data parallel algorithm could not exploit the task parallelism that is available in divide and conquer algorithms, and a simple task-parallel algorithm could not exploit the data parallelism that is available[13]. By contrast, nested parallelism accomplishes the ability to take a parallel function and apply it over multiple instances in parallel. For example having a parallel Fast Fourier Transform and then using it recursively to work several sub-sequences in parallel.

The significantly enhanced expressiveness of the nested data parallel paradigm originates greater demands upon the computational model that implements it. The main subject discussed is how to divide of processor groups to mach the change from data parallel to task parallel behavior of a algorithm and how to switch from parallel code to serial code when the group contains only one processor.

In this work we examine some performance requirements to obtain nesting and show briefly how two paradigms, *HPF* and *//c-MPI* attempt to satisfy them.

High Performance Fortran (HPF) is an informal language standard. Its aims are to simplify the programming of data parallel applications for distributed memory MIMD machines and supply the lack of portability of the resulting programs [7][8].
For a MIMD architecture, an HPF compiler transforms this program into an SPMD code by partitioning and distributing its data as is specified, allocating computation to processors according to the locality of the involved data, and inserting, if is necessary, data communications. Although HPF is a Data Parallel language, it provide task parallelism, therefore, the Nested Parallelism can be achieved.
HPF augment a standard Fortran 90 [9] program with directives. A directive is a structured Fortran comment that are distinguished by starting the characters 'HPF$' immediately after the comment character. A directive can specify the data distribution, define the abstract processor or implement task parallelism. In especial, the ON [5] directive allows the programmer to control the distribution of computations among the current active processors set. This directive don't change the active processors set, the callee inherits the caller's active processors.
In HPF approved extensions is legal to nest ON directives, if the set of active processors named by the inner ON directive is included in the set of active processors from the outer directive. The nesting of ON directives can be useful for expressing parallelism along different dimensions [5].

The MPI standard defines the user interface and functionality for a wide range of message-passing capabilities [12]. Since its completion in June of 1994, MPI has become widely accepted and used. Implementations are available on a range of machines.
The major goal of MPI, as with most standards, is a degree of portability across different machines. This topics is central but the standard will not gain wide usage if this was achieved at the expense of performance. A crucial detail is that MPI was carefully designed so as to allow efficient implementations. The design choices seem to have been made correctly, since MPI implementations over a wide range of platforms are achieving high performance, comparable to that of less portable, vendor-specific systems.
MPI allows or supports scalability, an important goal of parallel processing. Finally, MPI, as all good standards, is valuable in that it defines a known, minimum behavior of message-passing implementations. This relieves the programmer from having to worry about certain problems that can arise. One example is that MPI guarantees that the underlying transmission of messages is reliable. The user need not check if a message is received correctly.
Particularly for develop our experiments, we are using *La Laguna C++* [11], a set of macros and functions that extend MPI and PVM with the capacity for nested parallelism.

Although the performance of Data parallelism and Task parallelism languages has been demonstrated to be competitive for resolve commons parallels problems, performance of them is not optima when they try to resolve divide and conquer algorithms.
The obtained results on an particular example, parallel FFT algorithm, on HPF and //c-MPI confirm that the Nested parallelism is the natural model to apply when divide and conquer techniques are used.

## References


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

[2] Blelloch, G., Hardwick J., Sipelstien j., Zahga M. - *NESL user's manual* Technical report CMU-CS-93-114, School of computer Science . Carnegie Mellon Iuniversity. July, 1995.

[3] 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.

[4] 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.

[5] High Performance Fortran Forum - *High Performance Fortran Language Specification*. 1997

[6] Li, X., Lu, P., Schaefer, J., Shillington, J., Wong, P.S., Shi, H. - *On the Versatility of Parallel Sorting by Regular Sampling*. Parallel Computing, 19, pp. 1079-1103. 1993.

[7] Merlin J., Hey A. - An introduction to High Performance Fortran. University of Southampton. 1994.

[8] Merlin, J., Chapman, B. *High Performance Fortran 2.0*. VCPC University of Viena. 1997.

[9] Metcalf M., Reid J., - *Fortran 90 Explained*. Oxford University Press. 1990

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

[11] Rodríguez C., Sande F., Leon C. and García L. - *Extending Processor Assignment Statements*. 2nd IASTED European Conference on Parallel and Distributed Systems. Acta Press. 1998.

[12] Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J. - *MPI: The complete Reference*. Cambridge, MA: MIT Press, 1996.

[13] *The Design and Analysis of Parallel Algorithms*. Prentice-Hall (1989)