SUPPORTING NESTED PARALLELISM

González, J. A.[2], León C.[2], Piccoli F.[1], Printista M.[1], Roda, J.L.[2], Rodríguez, C.[2] and Sande, F.[2]

[1] Departamento de Informática.
Universidad Nacional de San Luis
Ejército de los Andes 950.
San Luis. Argentina
{mpiccoli, mprinti}@unsl.edu.ar

[2]Departamento de Estadística , Investigación Operativa y Computación
Universidad de La Laguna.
Tenerife. Spain

Abstract

Many parallel applications do not completely fit into the data parallel model. Although these applications contain data parallelism, task parallelism is needed to represent the natural computation structure or enhance performance. To combine the easiness of programming of the data parallel model with the efficiency of the task parallel model allows to parallel forms to be nested, giving Nested parallelism.

In this work, we examine the solutions provided to Nested parallelism in two standard parallel programming platforms, HPF and MPI. Both their expression capacity and their efficiency are compared on a Cray- 3TE, which is distributed memory machine. Finally, an additional speech about the use of the methodology proposed for MPI is done on two different architectures.

Keywords : Parallel Programming model, Nested Parallel model, Divide and Conquer technique.

1. Introduction

Many parallel programming models have been proposed, differing in their flexibility, task iteration mechanisms, task granularities, and support for locality, scalability, and modularity. Two standard models that can be taken to representing parallel computation are Data parallelism and Task parallelism [1].

Data parallelism is one of the more successful efforts to introduce explicit parallelism to high level programming languages. Data parallel programming is particularly convenient for two reasons. The first, is its ease of programming. The second is that it can scale easily to larger problem sizes. Several data parallel language implementations are available now[13],[14]. However, a perceived disadvantage of data parallelism is that it is only applicable to problems where a large set of data has to be uniformly operated, it is to say, monolithic problems. Hence, a set of independent sub-computations is strongly associated to a subset of these data. Such computations are inherently parallelizable, but each computation itself must be sequential.

The task parallel model achieves parallelism by using multiply threads of control, each getting part of the problem. Although the multiple threads of control have the disadvantage of to be more difficult to understand and use, a task parallel approach allows efficient implementations of irregular algorithms.

Nested parallel model is an extension of standard data parallel model, which includes the capability of nested parallel invocations. In this way, it combines the ability to apply a function in parallel to each element of a collection of data and the ability to nest such parallel calls [2],[3].

In this paper we examine the solutions provided by the two standard parallel programming platforms, *HPF* and *MPI,* comparing their efficiency on a Cray- 3TE, which is a distributed memory machine.

From the unlimited scope of applications that benefit from Nested parallelism, it has chosen the Divide and Conquer technique since it provides an excellent scenario for benchmarking. Both the general technique and the particular case that will be considered all along the paper are introduced in section 2. The two following sections describe in detail the expression of a Nested Parallel Fast Fourier Transform, exploiting both data and code parallelism in MPI and HPF. The fifth and sixth sections present the comparative study of the computational results and the conclusions.

2. Divide and Conquer as a test bed for Nested Parallelism Constructs

The divide and conquer approach is characterised by dividing the problems into sub-problems that are of the same structure as the larger problem. Further divisions into still smaller sub-problems 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 [9][11][15]. 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 [18]. By contrast, Nested parallelism accomplishes the ability to take a parallel function and apply it over multiple instances in parallel.

```
1  procedure pDC(x: problem; r: solution);
2  begin
3    if trivial(x) then conquer(x, r)
4    else
5    begin
6      divide(x, $x_0$, $x_1$);
7      parallel do pDC($x_0$, $r_0$) || pDC($x_1$, $r_1$));
8      combine(r, $r_0$, $r_1$);
9    end;
10 end;
```

Figure 1. General frame for a parallel divide and conquer algorithm

Let us consider the special case of the divide and conquer approach presented in Figure 1 where both the solutions *r* and the problems *x* have a vectorial nature. In such case there are opportunities to exploit parallelism not only at the task level (line 7) but also in the divide and combine subroutines (lines 6 and 8). Thus, data parallelism can be introduced by doing every processor in the current group to work in a subsection of the array *x* in the division phase (respectively a subsection of *r* in the combination phase).

As benchmark instance for this paper we will consider the Fast Fourier Transform (FFT) algorithm. However, the proposed techniques have been applied to other divide and conquer algorithms with similar results. Consider a sequence of complex numbers $a=(a[0], ..., a[N-1])$ of length *N*. The Discrete Fast Fourier Transform (DFT) of the sequence *a* is the sequence $A=(A[0], ..., A[N-1])$ given by $A[i] = \sum_{k=0..N-1} a[k]\, w^{ki}$, where $w = e^{2\pi\sqrt{-1}/N}$ is the primitive *n*th root of the unity in the complex plane. The following decomposition can be deduced from the definition:

$$A[i] = \sum_{k=0..N/2-1} a[2k]\, w^{2ki} + w^{i} \sum_{k=0..N/2-1} a[2k+1]\, w^{2ki}$$

From this formula, it follows that the DFT A of a can be obtained by combining the DFT B of the even components and the DFT C of the odd components of a.

3. Nested Parallelism in MPI

The MPI standard defines the user interface and functionality for a wide range of message-passing capabilities [17]. MPI, as all good standards, is valuable in that it defines a known, minimum behaviour of message-passing implementations. This relieves the programmer from having to worry about certain problems that can arise in the underlying transmission of messages

Although the natural way to express Nested parallelism in MPI [7] is through the use of communicators and the *MPI_Comm_split* function, *MPI_Comm_split* carries a considerable overhead since its execution implies a lot of communications.

Particularly for develop our experiments, we are using *La Laguna C* [16], a set of macros and functions that extend MPI and PVM with the capacity for Nested parallelism.

The code in Figure 2 shows a nested implementation of the FFT in *MPI* [8]. The algorithm assumes that the input vector a is replicated in the initial set of processors, while the resulting DFT A is delivered block distributed. Let us also assume that the number of elements N is larger than the number p of processors. Variable Np holds the quotient N/p, W is the vector containing the powers of the primitive n-th root of the unity and vector D is used as a temporary buffer during the combination.

The key point in the code is the use of the macro (line 9) we have called *PAR*. The call to $PAR(f_1, p_1, s_1, f_2, p_2, s_2)$ is expanded so that two subgroups of the current group of processors are generated. While the first one executes function f_1, the second does the same with function f_2. After the rejoin, the two subgroups exchange the results of their computations. For each group $i \in \{1,2\}$ this result is constituted by the s_i bytes pointed by p_i. This exchange is done in a pair-wise manner in such a way that the processors in one of the subgroups send in parallel their results to their corresponding partners in the other subgroups. This methodology can be straightforwardly expanded for non-binary divisions. The code of the procedure *seqFFT* (call at line 20) is simply the result obtained serialising the code in Figure 2.

In a previous work, we showed that the time taken by macro PAR when the division (and reunification) is performed using the alternative division technique proposed above is negligible compared with the times needed by the *MPI_Comm_split* version [7].

```
void parDandCFFT(Complex *A, Complex *a, Complex *W, unsigned Np,
           unsigned stride, Complex *D) {
Complex Aux, *pW;
unsigned i, size;

  if(NUMPROCESSORS > 1) {
   /* Division phase without copying input data */
    size = Np*sizeof(Complex);
    /* Subproblems resolution phase */
    PAR(parDandCFFT(A, a, W, Np, stride<<1, D), A, size,
      parDandCFFT(D, a+stride, W, Np, stride<<1, A), D, size);
    /* Combination phase */
    for(i = 0, pW = W+(Np*NAME*stride); i < Np; i++, pW += stride) {
      Aux.re = pW->re * D[i].re - pW->im * D[i].im;
      Aux.im = pW->re * D[i].im + pW->im * D[i].re;
      A[i].re += Aux.re;
      A[i].im += Aux.im;
    }
  }
  else
    seqFFT(A, a, W, N, stride, D);
}
```

Figure 2. MPI Nested parallelism

4. Nested Parallelism in High Performance Fortran

High Performance Fortran (HPF) is a formal 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 [12][13].

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 augments a standard Fortran 90 [14] program. The initial aim of the High Performance Forum meetings held during 1995 and 1996 was to expand High Performance Fortran 1.1 with capabilities such as enhanced data distributions, task parallelism and computation control, parallel I/O and directives to assist communication optimisation. The final decision was not to consider all these extensions as part of the new version HPF 2.0, but as "HPF 2.0 Approved Extensions" [10]. The expression meaning that, to be considered standard HPF 2.0, a compiler must provide full support for the HPF 2.0 features, but it is not required to support any of the Approved Extensions. The only HPF compiler compliant with version 2.0 approved extensions is ADAPTOR [4].

HPF increases a Fortran 90 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 called inherits the caller's active processors.

```
1  recursive subroutine FFT (R,N2,NAME,k,NP2)
2    implicit none
3    integer, parameter :: N = 1024*1024
4    integer, intent (in) :: N2, NAME, k, NP2
5    complex, dimension (0:N2-1), intent(out) :: R
6    complex, dimension (0:N-1) :: A
7    complex, dimension (0:N/2-1) :: W
8    common // A, W
9    complex, dimension (0:N2-1) :: B
10 !hpf$ processors Set(NP2)
11 !hpf$ distribute (block) onto Set(1:NP2) :: R
12 !hpf$ align with R :: B
13   integer :: S, k2
14
15   k2 = k*2
16   if (NP2 > 1) then
17 !hpf$ on (Set(1:NP2/2)), resident
18     call FFT(B(0:(N2/2)-1),N2/2,NAME,k2,NP2/2)
19 !hpf$ on (Set((NP2/2)+1:NP2)), resident
20     call FFT (B(N2/2:N2-1), N2/2, NAME+k, k2, NP2/2)
21     S = N2/2
22     R(0:S-1) = B(0:S-1) + B(S:N2-1)* W(0:N/2-1:k)
23     R(S:N2-1) = B(0:S-1) - B(S:N2-1)* W(0:N/2-1:k)
24   else
25     call seqIterativeFFT (R,NAME, N2,N2/2)
26   end if
27 end subroutine FFT
```

Figure 3. FFT: Nested parallelism in ADAPTOR HPF 6.1

Task parallelism is expressed in HPF using three new directives. These extensions, proposed in [10], are the ON, RESIDENT and TASK_REGION directives. The ON directive specifies the set of processors to perform a computation. Line 17 in Figure 3 specifies that only processors in the first half have to execute the call in line 18. The RESIDENT directive, used with the former, asserts that accesses to the specified objects within the scope of the ON directive are local. Finally, the

TASK_REGION ... END TASK_REGION directive defines a block of code in which it can be guaranteed that only the specified active processors of an execution task need to participate in its execution, and that the other processors can skip it.

At any time in the execution of a HPF statement there are a set of processors involved. Line 10 in Figure 3 declares the set of current active processors. The ON directive restricts the active processors to those named in its *home*.

In HPF approved extensions it 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. As in the MPI code, the input array *A* is replicated in each processor (lines 6 and 8) while the result *R* is block distributed (line 11). A difference with the MPI code is the subroutine called in the sequential case (line 25). The *seqIterativeFFT* procedure is the iterative solution used in the CMU Task Parallel Program suite [6].

5. Comparative Analysis

The experiences were carried out in CRAY 3TE, at Ciemat, Spain. This is a MIMD (*Multiple Instruction Multiple Data*) machine, massively parallel with distributed memory. It has 32 processor DEC EV-5 (Alpha), with 128M of main memory size. The MPI library was the CRAY native implementation. The HPF compiler was GMD ADAPTOR 6.1 installed on top of MPI.

Columns in Tables 1 and 2 present for the different software platforms, the running times and speed up respectively.

PROCS	HPF	MPI
1	11.970	5.95
2	6.748	3.03
4	3.298	1.58
8	1.673	0.81
16	0.835	0.42

Table 1. Running time to FFT. 1 Mega complex Cray- 3TE.

PROCS	SP-HPF	SP-MPI
2	1.77	1.96
4	3.63	3.76
8	7.15	7.34
16	14.33	14.16

Table 2. Speed Up to FFT. 1 Mega complex Cray- 3TE.

We also ported the MPI algorithm to a SGI Origin 2000. The SGI Origin 2000 used is a shared distributed memory machine with 64 MIPS R10000 processors and 8 GB of main memory. The results appear in Table 3.

PROCS	SGI-Origin 2000		Cray -T3E	
	Time	Speed Up	Time	Speed Up
1	9.091		5.95	
2	7.201	1.26	3.03	1.96
4	3.604	2.54	1.58	3.76
8	2.211	4.11	0.81	7.34

Table 3. FFT - 1 Mega complex, MPI implementation.

6. Conclusions

The purpose of Nested data parallel is provided the advantages of data parallelism while extending their applicability to algorithms that use ‘irregular’ data structures. The main advantages of data parallelism that should be preserved are efficient implementation of fine-grained parallelism and the simple synchronous programming model.

We have described a methodology to implement Nested Parallelism for Divide and Conquer algorithms. The results obtained prove that, not only the MPI performance is considerably better than the provided by HPF, but what is more remarkably, the effort invested in software development is similar.

7. Acknowledgements

We wish to thank to CCCC (Barcelona), CIEMAT (Madrid) for allowing us the access to their computational resources, like thus also to the National University of San Luis, CONICET, and ANPCYT from which continuous support is received

8. 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 luniversity. 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] Brandes, T.: ADAPTOR Programmer's Guide (Version 6.0). Technical Documentation, GMD, http://www.gmd.de/SCAI/lab/adaptor/adaptor_home.html (1998).

[5] Brandes, T.: Exploiting Advanced Task Parallelism in High Performance Fortran via a Task Library . Proceedings of Euro-Par'99 Parallel Processing, Toulouse, France. (1999).

[6] Dinda, P., Gross, T., O'Hallaron, D., Segall, E., Sttichnoth, E., Subhlok, J., Webb, J., Yang, B.: The CMU Task Parallel Program Suite. Technical Report CMU-CS-94-131, School of Computer Science, Carnegie Mellon University (1994).

[7] González, J.A., León, C., Piccoli, F., Printista, M., Roda, J.L., Rodriguez, C., Sande, F.. *Groups in Bulk Synchronous Parallel Computing.* Proc. 8th Euromicro Parallel and Distributed Processing. Pp 246-253. January 2000.

[8] Gonzalez, J.A., Leon, C., Piccoli, M.F., Printista, M, Roda, J.L., Rodriguez, C., Sande, F. *"Collective Computing".* V Congreso Argentino de Ciencias de la Computación en la Universidad del Centro. Octubre 1999. Tandil, Argentina.

[9] 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.

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

[11] 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.

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

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

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

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

[16] 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.

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

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