Dynamic Redistribution on Heterogeneous Parallel Computers

Dominique Sueur and Jean Luc-Dekeyser

Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille

Abstract. New generations of scientific codes trend to mix different types of parallelism. Algorithms are defined as a set of modules, with data parallelism inside modules and task parallelism between them. With high speed networks, tasks running on a heterogeneous computing environment can exchange data in a reasonable delay. Therefore data-parallel tasks distributed on different parallel computers can interact efficiently by reading or writing Data Parallel Objects. These objects are distributed on the physical nodes according to the mapping directives. Migrations of data parallel objects from one parallel computer to another lead us to define efficient algorithms for runtime array redistribution.
In this work, we have specially cared about the ability to handle distinct source and target processor sets while performing redistribution and the ability to overlap communications and computations. Performance results on a farm of *ALPHA* processors are discussed.

1 Introduction

Thanks to data parallelism, we can take advantage of parallel machines keeping an unique instruction stream to express parallelism. This programming model takes advantage of fine grain parallelism which can be found in numerous algorithms (matrix computations, image processing, ...). However, large applications often have an heterogeneous structure. A large grain parallelism insures the cooperation between few tasks, each task refers the data parallelism model to express the local parallel algorithm. With efficient networks like FDDI, HIPPI or ATM, it becomes possible to structure heterogeneous applications as a set of communicating data parallel tasks and to use the best architecture for each of them[2, 3].

To insure data parallel object migrations, we developed a command language called `Dpshell` [4]. Its goal is to build an heterogeneous application with data parallel program libraries. Programs are data parallel tasks which communicate by reading or writing Data Parallel Objects (DPO) by the way of `Dpshell` commands. Dependencies between data parallel modules are explicitly expressed at the command language level. The user can write his local data parallel algorithm on any machine and then combines the running programs to perform his entire application. Modules can then be reused to interactively create new applications. One of the characteristics of the `Dpshell` is to move the DPO from one parallel machine to another. From the processor point of view, this migration triggers a parallel Input/Output. The input (resp output) subroutine only needs to know the source (resp target) data distribution. It is the **Dpshell** responsibility to provide the elementary data migration according to the required distributions.

Implementation of parallel Input/Output between data-parallel tasks required dynamic redistribution over heterogeneous sets of processors. In this paper, we present a mathematical representation of $Cyclic(K)$ to $Cyclic(K')$ redistributions from a set of P processors to a distinct set of P' processors. Note that $Cylic(K)$ distributions include usual $Block(K)$ distributions. Based on this representation, we study special cases for which efficient algorithms can be identified. For each case we are able to overlap communications and computations. Performance results show that specialized algorithms are up to ten times faster than general algorithm.

2 Mathematical representation

Let T be an array of size N distributed $cyclic(K)$ over P processors. We visualize the layout of array elements in processor memories as sequences of blocks. Three values describe the location of an array element $T(i)$: $p = (i/K) \bmod P$ is the processor holding $T(i)$, $n = i/PK$ is the block index on processor p and $x = i \bmod K$ is the offset within the block. Therefore, we can write :

$$i = pK + nPK + x \tag{1}$$

with $x \in [0, K[$, $n \in [0, L(p)/K[$ and $p \in [0, P[$.

An heterogeneous redistribution involves two distinct sets of processor. The problem is to move an array T from one distribution D, $Cyclic(K)$ over P processors towards an other distribution D', $Cyclic(K')$ over P' processors. Let $E_D(p)$ be the set of indices of array elements hold by processor p with the source distribution D, and $E'_{D'}(p')$ the set of indices of array elements hold by processor p' with the target distribution D'. A redistribution routine needs to figure out exactly which data need to be sent between each processor pair. The processor p have to send to the processor p' the set $I(p,p') = E_D(p) \cap E'_{D'}(p')$. By using the equation (1) we can write :

$$\begin{aligned} i \in I(p,p') &\Longleftrightarrow i \in E_D(p) \cap E'_{D'}(p') \\ &\Longleftrightarrow pK + nPK + x = p'K' + n'P'K' + x' \ \bigwedge \left\{ \begin{array}{r} p \in [0, P[\\ n \in [0, L(p)/K[\\ x \in [0, K[\\ p' \in [0, P'[\\ n' \in [0, L'(p')/K'[\\ x' \in [0, K'[\end{array} \right. \end{aligned} \tag{2}$$

Solutions of this diophantine equation can be enumerated by using the general algorithm decribed in the next section.

3 General algorithm

In this enumeration approach, each source processor computes using equation (1) the index of each element it owns according to the source distribution. This index is then used to compute the target processor. The corresponding data item is packed into a buffer meant for that processor. When all address computations have been made data are sent to the target machine. Target processors perform the reverse job when they have received all their data. This algorithm is very simple and can be used for every *Cyclic*(K) to *Cyclic*(K') distributions, but it is very expensive because data migrations only takes place when all address computations have been achived. To take advantage of an heterogeneous environment and to overlap communications and computations, other algorithms have to be considered.

4 Specialized algorithms

To solve the diophantine equation (2) and to build efficient algorithms, we consider specific redistributions in which one or more variables can be fixed to zero. For these cases, we show that we can enumerate the solutions of the diophantine equation by using a set of regular sections $(first : last : stride)$. By using regular sections, a given processor p can create the $I(p, p')$ set meant for target processor p' without having to compute adresses of his whole local data. At each redistribution step, p creates one message and send it to the target machine. Computations in source and target processor sets can then be done in parallel and communications are overlapped with computations. All these specialized algorithms have the following properties :

The number of message sent is minimal: there is at most one message for each processor pair ;

Only required data are sent : only array items are sent (to be fully asynchrone we also send the source processor index) ;

Computations are dynamics : it is sufficient to know distribution parameters only at runtime ;

Constant time adress computations : the preprocessing time required to compute the regular sections only depends of distribution parameters ;

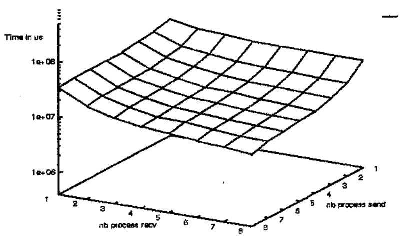
Computations are overlapped with communications ;

Two levels of parallelism are used : two data parallel tasks are runing concurently on the source and target computers.

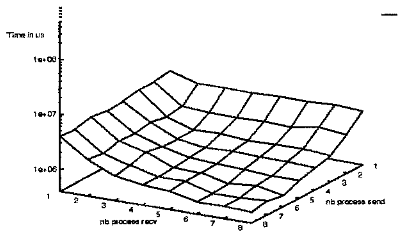
Figure (1) presents a global view of every studied case. In the next section performance results obtained with these specialized algorithms will be discussed.

Variables set to zero	Redistribution algorithms	Step of the regulars sections
n, n'	$Block(K)$ to $Block(K')$	$\Delta x = \Delta x' = \Delta p = 1$
n	$Block(K)$ to $Cyclic(K')$	$\Delta x' = \Delta n' = \Delta p' = 1$ $\Delta x = P'K'$
n'	$Cyclic(K)$ to $Block(K')$	$\Delta x = \Delta n' = \Delta p' = 1$ $\Delta x' = PK$
x'	$Cyclic(TK')$ to $Cyclic(K')$	$\Delta n = P'/GCD(PT, P')$ $\Delta n' = PT/GCD(PT, P')$ $\Delta p' = GCD(PT, P')$
x	$Cyclic(K)$ to $Cyclic(TK)$	$\Delta n = TP' * GCD(P, P'T)$ $\Delta n' = P/GCD(P, P'T)$ $\Delta p' = GCD(P, P'T)/GCD(T, D)$

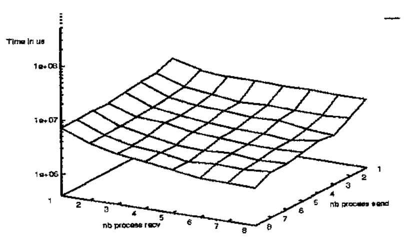
Fig. 1. Specialize algorithms



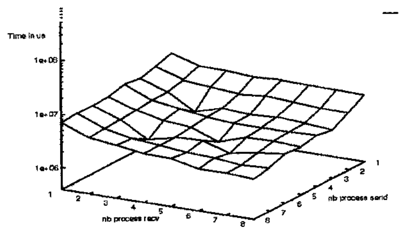
(a) Enumerative Algorithm



(b) $Block(K)$ to $Block(K')$



(c) $Block(K)$ to $Cyclic(K')$



(d) $Cyclic(TK')$ to $Cyclic(K')$

5 Experimental results

All experiments reported on this section have been done on a ALPHA farm of 16 processors interconnected by a cross-bar supporting FDDI bandwidth. We used an array of 4 Millions of integers. Algorithms were implemented over PVM3 library. In each case, we record the time necessary to redistribute the array T from a first set, S of P processors, to another set, T, of P' processors. We always keep $S \bigcap T = \emptyset$. Specialized algorithms always perform better than the enumeration algorithm because the redistribution is done in an overlapped fashion. In the *Block*(K) to *Block*(K') redistribution the difference is more accused as there is only one loop and all accesses are contiguous.

6 Conclusion

In this article, we proposed a mathematical representation of *Cyclic*(K) to *Cyclic*(K') redistributions. By using this representation, we resolved the corresponding diophantine equation and obtained algorithms which overlap communications and computations. These algorithms can be used in most usual cases. In heterogeneous environments, parallel I/O generaly required multiple redistributions. First data are moved from source compute nodes to source I/O nodes, then between source and target I/O nodes, and at last from target I/O nodes to target compute nodes. Strategies have to be developed to chose the inner distributions in order to minimized redistribution cost.

Complex scientific applications provide opportunities for exploiting multiple levels of parallelism. Thanks to our efficient algorithms, data-parallel tasks could exchange data in a reasonable delay. Note that these algorithms can also be used to implement HPF distribution and redistribution directives when there are several processor directives.

References

1. B. Avalani, A. Choudhary, I. Foster, R. Krihnaiyer, and M. Xu. A data transfer library for communicating data-parallel tasks. Technical report, Syracuse University, Argone Natinal Laboratory, NY 13244, 1994.
2. D. A. Carlson. Ultrahigh-performance FFTs for the CRAY-2 and CRAY-YMP supercomputers. *The journal of supercomputing*, 6(2):107–116, june 1992.
3. G. Edjlali, N. Emad, and S. Petiton. Hybrid methods on network of heterogeneous parallel computers. *Proceedings of the 14th Imacs world congress Atlanta, USA*, july 1994.
4. D. Sueur. Shell hétérogène à parallélisme de données. In *Renpar'7, Actes des 7 Rencontres Francophones du parallélisme*, pages 58–61, PIP-FPMs Mons, Belgique, June 1995.