Parallel Processing Letters, Vol. 11, No. 1 (2001) 41–56

EFFICIENT SUPPORT FOR SKELETONS ON WORKSTATION CLUSTERS

MARCO DANELUTTO

Dept. of Computer Science, University of Pisa, Corso Italia 40, I-56125 PISA, Italy
E-mail: marcod@di.unipi.it, http://www.di.unipi.it/~marcod

Received March 2001
Revised March 2001
Accepted by P. Evripidou & R. Perrot

ABSTRACT

Beowulf class clusters are gaining more and more interest as low cost parallel architectures. They deliver reasonable performance at a very reasonable cost, compared to classical MPP machines. Parallel applications are usually developed on clusters using MPI/PVM message passing or HPF programming environments. Here we discuss new implementation strategies to support structured parallel programming environments for clusters based on skeletons. The adoption of structured parallel programming models greatly reduces the time spent in developing new parallel applications on clusters. The adoption of our implementation techniques based on macro data flow allows very efficient parallel applications to be developed on clusters. We discuss experiments that demonstrate the full feasibility of the approach.

Keywords: **cluster computing, parallel programming models, skeletons, macro data flow**

1 Introduction

A computer cluster is basically a collection of complete computer systems interconnected by means of a local area network [1]. Beowulf class clusters [2], in particular, are clusters built using *commodity-off-the-shelf* components. Such kind of "parallel architectures" came into the scene some years ago as an alternative to classical MPP architectures. Cluster designers and users claim to reach performance values close to the ones provided by MPPs at a fraction of the cost of an MPP architecture on these systems. Clusters are conquering more and more positions in the Top500 [3] parallel architecture list. Actually, the top position in the last version of the list are maintained by the ASCI clusters [4] and SMP clusters occupy 112 positions of the list. Although these parallel machines cannot be classified as Beowulf clusters, mainly due to the adoption of high-end interconnection networks and to the number of nodes used, the point is that cluster parallel machines represent a rough 20% of the most powerful computer installations in the world.

Since long time, people made a distinction between networks and clusters of workstations (NOW vs. COW). A NOW is basically a cluster whose processing elements are used as normal workstations and, in addition, as processing elements of a cluster. Conversely, in a COW, the processing elements are dedicated to the

usage as cluster nodes. Therefore no other software than that needed to run cluster applications is run on such processing elements, nor they are used to run normal "production" application code, such as Office suites, WEB browsers and mailers. Most of the difference between COWs and NOWs, however, rely in the fact that NOW programming systems must take into account load balancing strategies at a deeper level than COW programming systems. It is normal, when working on a NOW, that a user coming back to his workstation automatically claims back all the processing power of the workstation for his personal purposes. At that point, NOW software must reallocate computations in such a way that his workstation is no more considered to belong to the cluster. Such subtle difference between NOWs and COWs make the NOW peak performance lower than the COW peak performance, in the general case, due to the additional load balancing/user tracking software needed to run parallel applications on NOWs. Notwithstanding this, network of workstations appear in 18 positions (about 4%) in the Top500 list mentioned above. As we are mainly interested in high performance, we focus our attention to COWs, although in Sec. 4 we describe the results of some experiments we run on both a COW and a NOW parallel architecture.

Most of the parallel applications running on clusters are developed using a restricted set of parallel programming environments, namely MPI [5], PVM [6] or some flavor of HPF [7]. Such programming environments either require the programmer to write code for all the details of the parallel application, such as communications, scheduling, mapping, etc., or they just provide some parallelism exploitation patterns. As an example, the HPF programming model just provides data parallel patterns. Therefore, any kind of control parallelism should be hand coded by the programmer, usually exploiting the operating system features (processes, threads, sockets and the alike). On the other hand, MPI or PVM programming libraries force the programmer to explicitly deal with all the process decomposition, scheduling and mapping details plus those details involved in inter-process communication setup. All these details require a deep knowledge of the underlying physical architecture features in order to be efficiently programmed. Furthermore, the efficient handling of these details definitely represents a non trivial, error prone programming effort.

In order to overcome such kind of problems, the current research trend is moving towards the adoption of new programming models, either based on some kind of skeleton based coordination language [8,9] or on the exploitation of the concept of ***design patterns*** [10,11,12], derived from the object oriented framework (see Section 2). In both cases the emphasis is on the development of ***structured*** parallel programming models that provide the programmer with simple ways of defining the structure of parallel applications by using either language constructs/skeletons or (parallel) pattern library entries.

Within this context, we summarize in this work some experiments we performed in the implementation of a structured parallel programming environment explicitly designed for Beowulf clusters. The experiences mainly concern the implementation of parallel programming environments based on skeletons [13] by using a dynamic run time support based on macro data flow (Sec. 3). We report the results (Sec.