Data abstractions for portable parallel codes

Javier Fresno*,
Arturo Gonzalez-Escribano*,
Diego R. Llanos*

* *Departamento de Informática, Edif. Tecn. de la Información, Universidad de Valladolid, Campus Miguel Delibes, 47011 Valladolid, Spain.*

ABSTRACT

Development of parallel software is a very complex task. Many details, such as domain type, partition method, or mapping scheme, should be considered during the implementation phase. However, they are usually taken into account in the design phase because the developers do not have the appropriate abstractions. This causes that small conceptual changes, for example a modification in the distribution policy, imply the developing of an application from scratch. The result is that it is not always possible to obtain portable and efficient parallel codes.

In this paper, we present a programming methodology that decouples data representation, partitioning, and mapping from the algorithmic and parallel strategy decisions. This allows to define explicitly all the design details with the appropriate abstractions. Postponing the low level parallel decisions to the implementation. To test this methodology, we have developed a library that offers these abstractions allowing to generate highly portable code. Our experimental results show that the use of this library allows to achieve similar performance as carefully-implemented manual code while reducing programming effort.

KEYWORDS: Parallel; Data partition; Mapping; Portable code

1 Introduction

Development of portable and efficient parallel software is not a trivial task. It usually involves dealing with low-level details about the particular architecture and the data structure, in order to adapt the implementations of the algorithms.

The most extended models for programming parallel machines are those based on message passing libraries such as MPI, or shared memory APIs like OpenMP. To write portable code, the programmer has to deal with all the non-trivial issues of the problem decomposition, distribution, and synchronization. Parallel languages such as UPC [CDC+99] or Chapel [CCD+11], ease the programming hiding the communication issues. However, efficient communications cannot be always automatically derived from generic codes.

[1]E-mail: {jfresno, arturo, diego}@infor.uva.es

Another common problem is the management of data structures with sparse domains. Although they are quite common, most programing languages only have a native support for dense arrays, forcing the programmer to manually manage the data with expensive programming effort or to use domain specific libraries. In both cases, it is not direct to reuse the code developed previously for dense data structures.

In this paper we present a programming methodology to solve these issues. Our approach is based on abstractions (data storage, virtual topologies, and layout plug-ins) that allow to define the synchronization and the parallel computation explicitly while hiding the implementation details.

Using these abstractions at the design phase allows to focus on the algorithm decisions defining where the data partition and communications between computational units needs to be made. Then, with the appropriate tools, the design solution can be straightforward implemented and the resulting code will be highly portable. Only small tunning is necessary when selecting the domain type or the particular data partition.

To test this methodology, we have developed Hitmap [GETFL13, FGEL13], a library designed to build parallel programs in terms of the previous abstractions. Hitmap programs are independent of the exact partition functions selected. It automatically adapts the data distribution and communications efficiently applying the particular techniques at run time.

The paper is organized as follows. Section 2 describes the hitmap usage. Section 3 shows experimental results. Finally, Section 4 concludes the paper.

2 Hitmap usage methodology

In this section, we discuss how a typical parallel program is developed using Hitmap. Hitmap proposes a programming methodology that follows a waterfall model with the phases shown in Fig. 1. Decisions taken at any stage only affect subsequent stages.

Design phase The developer designs the parallel code in terms of logical processes, using local parts of abstract data structures, and interchanging information across a virtual topology of unknown size. The first step is to select the virtual topology type appropriate for the particular parallel algorithm.

The second design step is to define domains using a global view approach. All logical processes declare the shapes of the whole data structures used by the global computation. The developer chooses where to activate the partition and mapping procedure for each domain. At this phase, the only information needed is the domain to be mapped. There is no need to specify the particular partitioning technique to be used.

Local domains may be expanded to overlap other processors subdomains, generating *ghost zones*, a portion of the subspace shared with another virtual processor. Once mapped, and after the corresponding memory allocation, the programmer can start to use data in the local subdomain. The developer finally designs which communication structures are needed to synchronize data between the computational phases.

Implementation phase Hitmap provides functionalities to directly translate the design to an implementation which is independent of the underlying physical topology, and the partition/layout techniques. Hitmap provides several topology plug-ins that arrange the physical processors to build different neighborhood relationships.

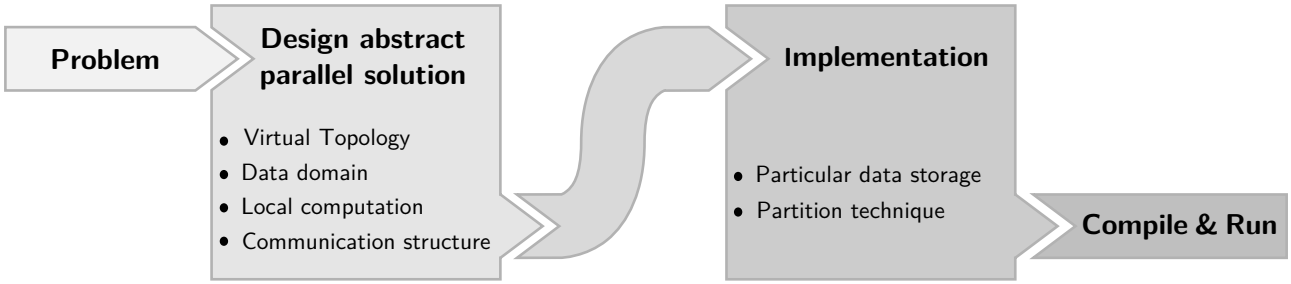

Figure 1: Phases of the hitmap methodology.

Layout objects are instantiated for partitioning and mapping global domains across the virtual topology. The layout objects are queried to obtain the local subdomain shapes. After expanding or manipulating the shapes if needed, the local tiles can be dynamically allocated. Communication objects are built to create the communication structures designed. They are instantiated using a local tile and using information contained in a layout object about neighbors and domain partition.

The result of the implementation phase is an generic code that is adapted at run-time depending on: (a) the particular global domains declared; (b) the internal information about the physical topology; and (c) the selected partition/layout plug-ins. Note that it is possible to change the partition name technique without affecting the rest of the code.

3 Experimental results

In this section, we compare the Hitmap version of two benchmarks with their respective handwritten versions to test the performance achieved. The handwritten versions are programmed in C and use MPI for communications. The first benchmark is the well know NAS MG, a structured multigrid kernel to solve a discrete Poisson problem. The second benchmark performs a neighbor-vertices synchronization in a sparse graph structure. It traverses the graph nodes, updating each node value using a function on the neighbor node values.

The codes have been run on two different architectures. The first one, Geopar, is an Intel S7000FC4URE server, equipped with four quad-core Intel Xeon MPE7310 processors at 1.6GHz and 32GB of RAM. The second architecture is a homogeneous Beowulf cluster of up to 36 Intel Pentium IV nodes, interconnected by a 100Mbit Ethernet network. Fig. 2 shows the execution times for both the handwritten and the Hitmap benchmark implementations. There is no significant performance difference between the two implementations. Therefore, the abstractions introduced by Hitmap do not lead to significant performance penalties.

4 Conclusions

In this paper we have presented Hitmap, a library for hierarchical tiling and mapping of dense arrays and sparse structures. Hitmap abstractions allow to develop explicit parallel programs that automatically adapt their synchronization and communication structures to dense or sparse data domains, and their specific partition/mapping techniques. The use of Hitmap leads to more abstract programs, easier to code and maintain, still obtaining good performance results.

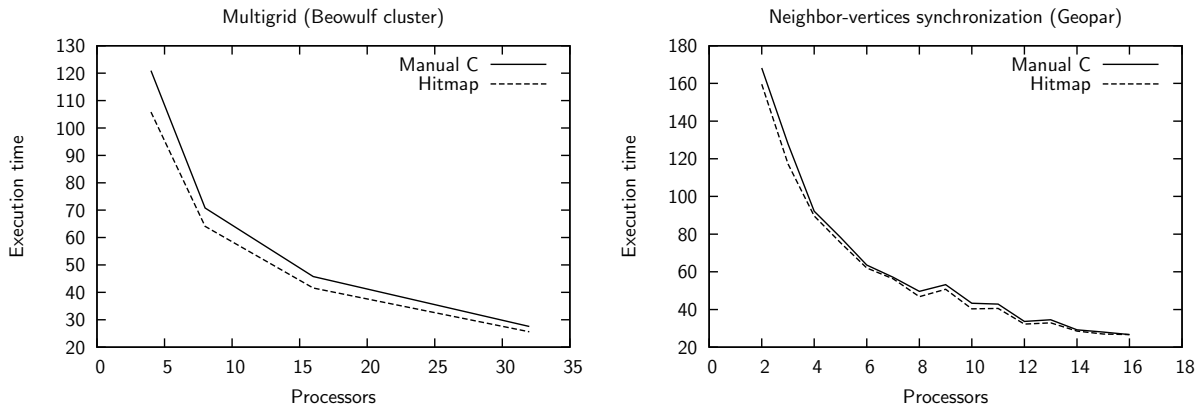

Figure 2: Execution time for manual and Hitmap implementations of the benchmarks.

Acknowledgments

This research is partly supported by the Castilla-Leon Regional Government (VA172A12-2); Ministerio de Industria, Spain (CENIT OCEANLIDER); MICINN (Spain) and the European Union FEDER (Mogecopp project TIN2011-25639, CAPAP-H3 network TIN2010-12011-E, CAPAP-H4 network TIN2011-15734-E); and the HPC-EUROPA2 project (project number: 228398) with the support of the European Commission - Capacities Area - Research Infrastructures Initiative, and the ComplexHPC COST Action.

References

[CCD+11] Bradford L. Chamberlain, Sung-Eun Choi, Steven J. Deitz, David Iten, and Vassily Litvinov. Authoring User-Defined Domain Maps in Chapel. Technical report, Cray User Group, 2011.

[CDC+99] William W. Carlson, Jesse M. Draper, David E. Culler, Kathy Yelick, Eugene Brooks, and Karen Warren. Introduction to UPC and Language Specification. Technical report, IDA Center for Computing Sciences, 1999.

[FGEL13] Javier Fresno, Arturo Gonzalez-Escribano, and Diego R. Llanos. Extending a hierarchical tiling arrays library to support sparse data partitioning. *The Journal of Supercomputing*, 64(1):59–68, April 2013.

[GETFL13] Arturo Gonzalez-Escribano, Yuri Torres, Javier Fresno, and Diego R. Llanos. An extensible system for multilevel automatic data partition and mapping. *IEEE Transactions on Parallel and Distributed Systems*, to appear, 2013.