Targeting Heterogeneous Architectures in ASSIST: Experimental Results*

M. Aldinucci[2], S. Campa[1], M. Coppola[2], S. Magini[1], P. Pesciullesi[1], L. Potiti[1], R. Ravazzolo[1], M. Torquati[1], and C. Zoccolo[1]

[1] Dept. of Computer Science – University of Pisa – Viale Buonarroti 2, Pisa, Italy
[2] Inst. of Information Science and Technologies – CNR, Via Moruzzi 1, Pisa, Italy

Abstract. We describe how the ASSIST parallel programming environment can be used to run parallel programs on collections of heterogeneous workstations and evaluate the scalability of one task-farm real application and a data-parallel benchmark, comparing the actual performance figures measured when using homogeneous and heterogeneous workstation clusters. We describe also the ASSIST approach to heterogeneous distributed shared memory and provide preliminary performance figures of the current implementation.

Keywords: Structured parallel programming, heterogeneous workstation network, shared memory.

1 Introduction

A notable problem when dealing with parallel programming environments is the ability to produce code for heterogeneous networks/clusters of workstations. Although some versions of MPI (e.g. LAM-MPI) allow heterogeneous collections of PEs to be used, other versions do not support such feature, nor it is supported by most of the other parallel programming systems, including HPF.

ASSIST is a parallel programming environment based on the concepts of coordination languages and algorithmical skeletons, recently developed at the University of Pisa [1, 2]. The latest version (1.2) of the ASSIST programming environment supports both interoperability with other classical distributed-processing frameworks (e.g. CORBA) and the possibility to run portions of the same application on machines with different processors and operating systems.

The former is presented in [3], so this paper presents the latter. In Sect. 2 we discuss the current implementation of point-to-point communication and shared memory in an heterogeneous environment, highlighting strengths and weaknesses. In Sect. 3 the performance obtained running existing ASSIST applications and benchmarks on heterogeneous platforms are evaluated and compared with those obtained on homogeneous ones. Sect. 4 surveys related work.

* This work has been supported by the Italian MIUR FIRB Grid.it project, n. RBNE01KNFP, on High-performance Grid platforms and tools; the Italian MIUR Strategic Project L.449/97-2000, on High-performance distributed enabling platforms.

M. Danelutto, D. Laforenza, M. Vanneschi (Eds.): Euro-Par 2004, LNCS 3149, pp. 638–643, 2004.

2 Heterogeneous Network/Cluster Targeting

The ASSIST compiler can produce code running onto an heterogeneous cluster/network, composed of processing elements with different processors (word size, endianness) and/or different operating systems. In the experiment section we will show that the incurred overhead is tolerable, therefore efficiency and performance figures are preserved w.r.t. the homogeneous case.

The compilation process has been designed [2, 4] to generate code for heterogeneous clusters/workstation networks[1]. When compiling for an heterogeneous platform, the ASSIST compiler produces a full set of object files for every configured architecture, and a global configuration file, that can be used to start the application on a set of heterogeneous computing nodes, by means of a simple invocation of the ASSIST loader. In an heterogeneous run, processes participating in the computation of the same ASSIST-CL parallel pattern (e.g. task farm, pipeline, parmod, ...) can be scheduled on processing elements having different CPU/operating system combinations, with minimal performance impact.

The ACE library [5] provides an hardware and operating system abstraction layer to the ASSIST runtime. It also provides standard routines, based on the CDR format (a flavor of XDR), to exchange data between processing elements with different architectures, preserving their original semantics independently of the endianness or of the machine word size used on the different machines. The provided routines handle all CORBA basic types (octets, integers, floating point values, ...) as well as unidimensional arrays of basic types. The ASSIST compiler inductively builds the conversion routines for more complex data structures (structured types, multidimensional arrays), using the ones for the basic types. This process (unlike in MPI), is done at compile time, and exploits the inlining facility of the native C++ compiler, so the produced conversion routines are quite fast. The CDR protocol prescribes that the sender doesn't encode data, but enriches the message with its byte-order. The receiver compares the sender byte-order with its own, and if they are different, applies the conversion.

The CDR based approach incurs only one significant overhead: since the memory layout of the data structures that must be communicated can be different on different architectures even if the byte-order is the same (due to word-size or alignment requirements), the protocol defines an architecture independent layout, that can be interpreted on every supported architecture. So data structures must be copied when sending and when receiving, even if byte-order conversion is not needed. This enlarges the memory footprint of the parallel program (a noticeable effect if the communicated structures are large), and can produce cache pollution effects.

We are currently working on an heterogeneity-enabled version of the shared memory library integrated in ASSIST [6]. The library provides the ASSIST programmer with a set of primitives that allow to allocate, read, write and deallocate

[1] Although at the moment only Linux and MacOS X machines has been configured and tested, the ASSIST compiler and runtime support already contains all the hooks needed to target other architectures/operating systems (namely the ones supported by the Hardware Abstraction Layer, based on the ACE library).

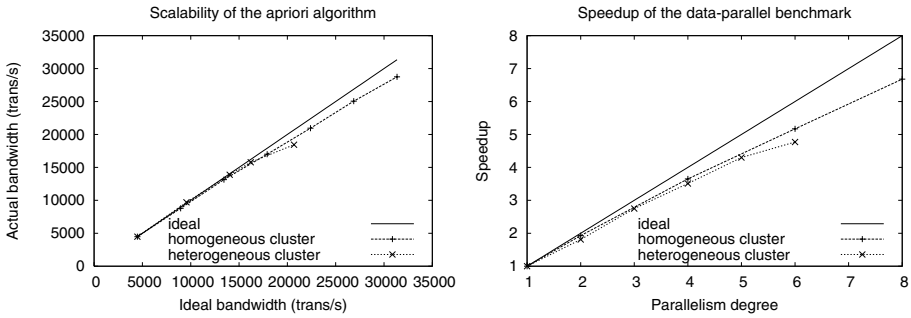

Fig. 1. Comparison of homogeneous and heterogeneous performance figures.

segments of distributed shared memory. Types are necessary in order to select the proper conversion routines, so we defined a typed API for the shared memory operations, exploiting the C++ template mechanism. In the implementation we reused the same conversion routines built for inter-process communication, but with a different philosophy: in the interaction through shared memory, the memory itself plays the role of a third party between the interacting entities, unaware of the data type that is being exchanged. Moreover, the receiver doesn't know the identity of the sender (it can happen that different entities write different entries in one array and a single entity read the array as a whole). To arrange for this, we decided to always encode data written to the shared space in an external data format, with a well defined layout and byte-order, and symmetrically decode data read from the memory. The chosen layout guarantees that all the elements of an array have the same layout (surprisingly, this is not true for plain CDR layout, when dealing with structured data types), in order to be able to read/write portions of arrays.

3 Performance Evaluation

We are going to show experimental results concerning application scalability, comparing the performance figures obtained with an homogeneous platform with those obtained with an heterogeneous one.

Environment. The homogeneous tests are run on a cluster of 8 Pentium IV 2GHz, equipped with 512MB of RAM and interconnected by 100/1000 Mbit/s Ethernet.

The heterogeneous tests are run on a cluster of 4 Pentium IV (same configuration), a fast PowerPC G4 (1.5GHz, 512MB RAM, 100/1000 Mbit/s Ethernet) and a slower one (800MHz, 256MB RAM, 100 Mbit/s Ethernet).

Apriori algorithm. The first algorithm we tested is a parallel implementation the Apriori data mining algorithm [7] for finding association rules in a transaction database. The transaction database is split in a stream of partitions that feeds

a task-farm; the farm workers compute the frequent itemset independently in each partition, and then the partial results are accumulated to build a superset of the solution; in a second parallel scan of the database (another task-farm working on the stream of partitions) the partial solutions are ranked and a complete solution is found. In this implementation, the database partitions are sent over the ASSIST streams, to exploit the data conversion mechanisms between heterogeneous architectures.

The scalability for the homogeneous and the heterogeneous executions are compared in figure 1-left. The heterogeneous machines have different computational power: the Pentiums can process 4478 transactions per second, while the fast and the slow PowerPC can process respectively 5079 and 2174 transactions per second. The heterogeneous configurations tested are all the prefixes of the sequence [Pentium, fast PPC, Pentium, slow PPC, Pentium]. For different heterogeneous configurations, we computed the ideal bandwidth (x axis) as the sum of the bandwidths of the machines employed, and measured the delivered bandwidth (y axis). The curve obtained for the heterogeneous case is comparable with the homogeneous one. It departs from the ideal (reaching 0.9 of efficiency) only when the machine with slow NIC is employed: the application, in fact, has high network bandwidth requirements, and the introduction of a node with a slow network slightly decreases the efficiency of the task distribution process.

Data-parallel benchmark. The second algorithm tested is a synthetic data-parallel benchmark, with a variable communication stencil and featuring a good computation to communication ratio The benchmark implements an iterative computation over a square matrix M: at iteration h the h^{th} row of M is broadcast to all the processing elements; the new value for the matrix M' is computed as $M'_{i,j} = N \cdot sin(\sum_k M_{h,k} \cdot M_{j,(j+k) \bmod N})$.

The ASSIST support currently implements only a naive partitioning strategy for data-parallel computations, in which all the partitions have the same size, even if the computation power of the employed machines differs. We can therefore compute the speedup of the program against the sequential time of the slowest machine. The sequential times for Pentiums, fast PPC and slow PPC were respectively 529, 113 and 197 seconds. The difference in the execution times between the Pentiums and the PowerPCs is considerable: the Pentiums are disadvantaged in number crunching codes because of the small number of general registers; register pressure, in fact, prevents the compiler from optimizing complex matrix access patterns.

The heterogeneous configurations tested are all the prefixes of the sequence [Pentium, fast PPC, Pentium, slow PPC, Pentium, fast PPC again]; in the maximal configuration we mapped two workers on the fastest machine, that otherwise would be idle most of the time. Figure 1-right displays the speedups obtained in homogeneous as well as heterogeneous runs. The overhead introduced by heterogeneity is negligible (less than 5%). The inability to adjust the partition sizes proportionally to the machine powers, instead, is limiting; moreover the solution of running more workers on faster machines introduces some inefficiencies (the heterogeneous speedup curve, in fact, loses linearity when the second worker is

added). We are now considering to enhance the partitioning strategy to handle computational power heterogeneity, as well as dynamic changes in the available computational power.

Shared memory performance. Here we provide the first performance results regarding the described implementation of the heterogeneous shared memory library integrated in ASSIST. The benchmark employed allocates one segment of shared memory (10M integers); several processes in parallel read randomly chosen chunks (16k integers) of this memory and write them in other locations. The accesses are not synchronized. Figure 2 shows the aggregate bandwidth of the shared memory varying the parallelism degree (number of servers) on the 100Mbit/s network, both in homogeneous runs and in heterogeneous ones (the heterogeneous configuration has been enriched with two more Pentium IV machines, hosting two memory servers). The overhead introduced is always less than 6%, and decreases when the number of servers increase.

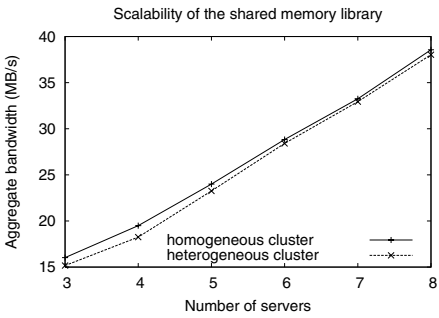

Fig. 2. Performance of the shared memory library ($\#clients = \#servers - 2$).

4 Related Work

PBIO [8] is a flexible communication library that handles heterogeneous communications using sender's native data representation: the receiver translates the message only if necessary, basing the decision on a message prefix describing the data format. Our approach, instead, adopts a consistent data layout, that is useful when we extend to shared memory. Mermaid [9] is the first example of an heterogeneous, transparent page-based DSM; in this scheme data is represented in the page holder's native form: this introduces several difficulties and limitations in the architectures that can be supported. Heterogeneous DSMs can benefit of type safe and reflective languages like Java. JavaDSM [10] is a prototypical implementation of a DSM that offers a single JVM image over a cluster of possibly heterogeneous workstations.

5 Conclusion

In this paper we described ASSIST support to heterogeneity, i.e. the ability to produce code for heterogeneous networks/clusters, and run portions of the same application (and even of the same parallel pattern) on machines with different processor architectures and operating systems.

We discussed the implementation of point-to-point communications, highlighting its strengths and weaknesses; we provide experimental evidence that

this approach incurs in tolerable overhead, comparing the performance figures obtained by heterogeneous runs to those obtained by homogeneous ones of a real task-parallel program (the Apriori data mining algorithm) and a synthetic data-parallel benchmark.

Finally, we extended the approach to the shared memory library integrated in ASSIST and presented first experimental results showing that the impact of data translation on the achieved performance is small.

Acknowledgements

We wish to thank all the people that participated to the design and development of the ASSIST programming environment, in particular P. Ciullo, M. Danelutto, G. Giaccherini, A. Paternesi, A. Petrocelli, E. Pistoletti, P. Vitale, M. Vanneschi, G. Virdis.

References

1. Vanneschi, M.: The programming model of ASSIST, an environment for parallel and distributed portable applications. Parallel Computing **28** (2002) 1709–1732
2. Aldinucci, M., Campa, S., Ciullo, P., Coppola, M., Magini, S., Pesciullesi, P., Potiti, L., Ravazzolo, R., Torquati, M., Vanneschi, M., Zoccolo, C.: The Implementation of ASSIST, an Environment for Parallel and Distributed Programming. In Kosch, H., László Böszörményi, Hellwagner, H., eds.: Euro-Par 2003: Parallel Processing. Number 2790 in Lecture Notes in Computer Science (2003) 712–721
3. Magini, S., Pesciullesi, P., Zoccolo, C.: Parallel Software Interoperability by means of CORBA in the ASSIST Programming Environment. In: Euro-Par 2004 Parallel Processing. (2004) (to appear).
4. Aldinucci, M., Campa, S., Ciullo, P., Coppola, M., Danelutto, M., Pesciullesi, P., Ravazzolo, R., Torquati, M., Vanneschi, M., Zoccolo, C.: A Framework for Experimenting with Structured Parallel Programming Environment Design. In: ParCo 2003 Conference Proceedings, to appear, Dresden, Germany (2003)
5. Schmidt, D.C., Harrison, T., Al-Shaer, E.: Object-oriented components for high-speed network programming. In: Proceedings of the 1st Conference on Object-Oriented Technologies and Systems (COOTS), Monterey, CA, USENIX (1995)
6. Carletti, G., Coppola, M.: Structured Parallel Programming and Shared Objects: Experiences in Data Mining Classifiers. In Joubert, G.R., Murli, A., Peters, F.J., Vanneschi, M., eds.: Parallel Computing, Advances and current issues, Proceedings of the Int. Conf. ParCo 2001, Naples, Italy (4-7 September 2001), Imperial College Press (2002) ISBN 1-86094-315-2,.
7. Coppola, M., Vanneschi, M.: High-Performance Data Mining with Skeleton-based Structured Parallel Programming. Parallel Computing, special issue on Parallel Data Intensive Computing **28** (2002) 793–813
8. Eisenhauer, G., Bustamante, F.E., Schwan, K.: Native Data Representation: An Efficient Wire Format for High-Performance Distributed Computing. IEEE Transactions on Parallel and Distributed Systems **13** (2002) 1234–1246
9. Zhou, S., Stumm, M., Li, K., Wortman, D.: Heterogeneous Distributed Shared Memory. IEEE Trans. on Parallel and Distributed Systems **3** (1992) 540–554
10. W.Yu, A.Cox: Java/DSM: A Platform for Heterogeneous Computing. In: Proc. of ACM 1997 Workshop on Java for Science and Engineering Computation. (1997)