Block Matrix Multiplication in a Distributed Computing Environment: Experiments with NetSolve[*]

Luisa D'Amore, Giuliano Laccetti, and Marco Lapegna

Department of Mathematics and Applications,
University of Naples Federico II,
via Cintia Monte S. Angelo, 80126 Naples, Italy
{damore, laccetti, lapegna}@dma.unina.it

Abstract. We address synchronization issues of some block matrix multiplication algorithms in a distributed computing environment. We discuss performance behavior of a client/server implementation of these algorithms focusing on the most appropriate version which delivers the minimum synchronization overhead. Numerical experiments are carried out using the NetSolve distributed computing system.

1 Introduction

Distributed computing aggregates computational resources in order to tackle problems that cannot be solved on a single system. Depending on the software and hardware infrastructure, these resources might comprise the majority of the supercomputers in the country or simply all the workstations within a department. Over the last decade, distributed computing received great attention from the scientific computing communities, thanks to a substantial improvement of the networks bandwidth, and it is now feasible the use of geographically scattered computers as a single computational resource. A significant example is the Italian national research network GARR [7], where the backbone bandwidth has grown from 2Mbit/sec in 1994 up to 2.5Gbit/sec in 2002, a growth factor of about 1000 in eight years, much more than the Moore's law about the processors speed. In general, for computationally demanding problems the exploitation of parallelism is a key issue in order to compute the solution within a reasonable time. This means to decompose the given problem into smaller subproblems to be solved concurrently (data or task decomposition).

However, it is important to underline that distributed computing environments are composed by heterogeneous computational resources, both from the static (processors, operating systems, arithmetic,...) and dynamic (workload of the systems, effective bandwidth of the networks,..) point of view, making very

[*] This work has been partially supported by Italian Ministry of Education, University and Research (MIUR) within the activities of the WP9 workpackage *Grid Enabled Scientific Libraries*, coordinated by A. Murli, part of the MIUR FIRB RBNE01KNFP Grid.it project.

difficult an efficient synchronization among the nodes. For this reason, one of the main approaches towards the development of distributed applications is based on the client/server programming model where the application is divided into a large number of essentially independent tasks that are dispatched to several servers, and a "coordinator" task managed by the client module. In the parallel computing community these problems are called "pleasingly" or "embarrassingly" parallel.

The most significant example in this sense is the SETI@home project [10]. This project uses idle computers for the analysis of radio signals outcoming from space to find extraterrestrial intelligence, and it has been able to aggregate more than 5 millions of computers, achieving a average performance of about 60 Tflops, about the same of today's most powerful supercomputers [4].

Beyond such example of mere networked computing, new issues arise in the development of algorithms and software able to run efficiently on such a pervasive hardware and software infrastructure. Following [5], in this paper we deal with on-demand computing where remote resources are used to meet short term requirements that cannot be cost effectively or conveniently locally located. These resources may be hardware, software libraries, data repositories, and so on. In contrast to traditional supercomputing, these applications are often driven by cost-performance concerns rather than absolute performance. The challenging of on demand computing derives primarily from the dynamic nature of resources requirements and the potentially large populations of users and resources. Main issues include middleware related topics like resources brokering and management, configuration, authentication and security as well algorithm related aspects like fault tolerance, latency tolerance, heterogeneity management and performance.

In this paper our reference computational environment agrees with a client/server programming model where:

- the servers do not communicate directly each other but with the client only,
- the selection of the resources is in charge to the underlying computing environment by means of their own dynamic allocation strategies.
- the computational information about the servers, including their availability, load, processor speed, are hidden to the client.

In these years several computing environments have been developed with the aim to address these topics, allowing, at the same time, a friendly access to remote resources. Among them there are NetSolve [1] and Condor [8]. NetSolve is the distributed computing environment where we implemented our algorithms. It has been developed at the University of Tennessee to be a simple-to-use middleware system that allows users to access computational resources and to use remote libraries, without the need to locate, configure and install them. However, for the development of algorithms for distributed computing environments, a completely different approach with respect to the classical parallel computing methodologies, is necessary. As case study we consider a block matrix multiplication algorithm because it is a basic linear algebra computational kernel representative of similar other computations. On the other hand, it encompasses a lot of data movements, so that to minimize the synchronization overhead among the nodes becomes a challenging task.

The paper is organized as follows: in section 2 we analyze the performance behavior of some versions of block matrix multiplication algorithm in a client/server programming model, in section 3 we describe experiments on two different NetSolve systems: a “wide area” system located at the University of Tennessee and a “local area” system installed in our university campus. Finally, conclusions are discussed in section 4.

2 A Client Server Block Matrix Multiplication Algorithm

Starting from PUMMA (Parallel Universal Matrix Multiplication Algorithm) [3], the algorithm on which PDGEMM routine of ScaLAPACK is based, similar algorithms for block matrix multiplication have been developed in the last decade. These algorithms are tuned in order to optimize the synchronization overheads on tightly coupled distributed memory machines by overlapping computations and communications; recently, new issues on heterogeneous networks of workstations have been addressed; in this case the load balancing of processors running at different speed is the challenging task (e.g. [2] and [9]). These implementations are based on the revision of classical parallel algorithms for homogeneous environments and they suppose that key features of the computing resources, as the cycle times of the CPUs, are known, so it is possible to dispatch to each node an amount of work proportional to its computational speed. However, these algorithms are based on the SPMD programming model, suitable for distributed memory multiprocessors with a tightly intra node synchronization but critically liable for the performance decline in a client/server distributed computing environment.

As an example of revision for the classical parallel block matrix multiplication algorithms let us assume that A is an $(m \times n)$ matrix, B is an $(n \times p)$ matrix and C is an $(m \times p)$ matrix divided for simplicity of notations in square blocks of order r, with n, m and p divisible by r. Then the blocks number of the matrix are $MB = m/r$, $NB = n/r$ and $PB = p/r$.

In Figure 1, three variants of a standard blocks algorithm for the matrix multiplication $C = A \times B$, obtained by the permutation of the loops indices are

```
for I=1, MB (in parallel)
  for J=1, PB (in parallel)
    for K=1, NB
      C(I,J)=C(I,J)+
      A(I,K)B(K,J)
    endfor
  endfor
endfor
```

a) (I, J, K) ordering

```
for I=1,MB (in parallel)
  for K=1, NB
    for J=1, PB (in parallel)
      C(I,J)=C(I,J)+
      A(I,K)B(K,J)
    endfor
  endfor
endfor
```

b) (I, K, J) ordering

```
for K=1, NB
  for I=1, MB (in parallel)
    for J=1, PB (in parallel)
      C(I,J)=C(I,J)+
      A(I,K)B(K,J)
    endfor
  endfor
endfor
```

c) (K, I, J) ordering

Fig. 1. Standard versions of parallel blocks matrix multiplication

then shown. Note that the other versions are equivalent to these ones and all versions are based on the same matrix operation:

$$C(I,J) = C(I,J) + A(I,K)B(K,J) \tag{1}$$

In a client/server implementation, for given values of I, J and K, this operation can be computed by sending from the client to a server the three blocks $A(I,K)$, $B(K,J)$ and $C(I,J)$, then the server can update the block $C(I,J)$ and it can sends back the result to the client. It is important to note that in all cases the only possible source of parallelism is along the indices I and J, in the sense that each block $C(I,J)$ can be computed independently from the other ones. This is not possible for the index K, because of the risk of "race condition" accessing a given block $C(I,J)$ for different values of K. Then, in order to reduce the synchronization overhead the main problem in a client/server implementation is to define which of the orderings in Figure 1 has to be used to compute the several matrix operations involving the blocks $C(I,J)$, $A(I,K)$ and $B(K,J)$. For the (I,J,K) ordering (Figure 1.a) the client generates $MB \times PB$ independent threads of computation, each of them managing the sequence along the index K. For the (I,K,J) ordering (Figure 1.b) the client generates only MB independent threads of computation, each of them generating PB parallel tasks at every step of the index K . Finally, for the (K,I,J) ordering (Figure 1.c) at each step of the index K, the client generates $MB \times PB$ parallel tasks, and it has to wait for the completion of all these tasks before generating new ones.

For a computational cost analysis, let t_{ijk} denote the execution time (computation and communication) needed to perform the operation (1) and $T^{(a)}$, $T^{(b)}$ and $T^{(c)}$ the total execution times for the three orderings in Figure 1. It is easy to found that:

$$T^{(a)} = \max_{i,j} \sum_k t_{ijk} \quad T^{(b)} = \max_i \sum_k \max_j t_{ijk} \quad T^{(c)} = \sum_k \max_{i,j} t_{ijk}$$

so that:

$$T^{(a)} \le T^{(b)} \le T^{(c)}$$

Then the (I,J,K) ordering is more oriented to a distributed client/server implementation than the others two orderings. The worst case is the (K, I, J) ordering. Let us assume, for a while, the ideal case, where the environment is homogeneous and dedicated to the computation and $m = n = p$. In this case the execution time $t_{ijk} = t$, is the same for all the values of I, J, K, and

$$T^{(a)} = T^{(b)} = T^{(c)} = NB \cdot t$$

This result shows a linear growth with respect to NB of the total execution time when the matrix dimension n grows while the block dimension $r = n/NB$ is kept constant, and that the ideal scaled efficiency when we multiply by α the matrix dimension n is:

$$S_\alpha = T_n^{(a)}/T_{\alpha n}^{(a)} = \frac{n \cdot t/r}{\alpha n \cdot t/r} = \frac{1}{\alpha}$$

Finally let us compute the communication overhead of each task. In all the cases, for each value of K, the core instruction (1) is executed $MB \cdot NB \cdot PB$ times, each requiring 4 blocks communications of size r^2 ($C(I,J)$, $A(I,K)$, $B(K,J)$ and back the update of $C(I,J)$) between the client and the servers. This means that $4r^2$ double precision floating point data, are exchanged between the client and the servers. Then, the total communication overhead is due to the movement of

$$CO = \frac{32n^2 MB \cdot PB}{NB} \quad \text{byte.}$$

We implemented the three versions of the block matrix multiplication in the NetSolve distributed computing environment. This environment uses a client-agent-server paradigm aimed to dispatch on the most suitable server those calculations that require specialized software and, at the same, to minimize the total communication overhead among the nodes of the environment. More precisely, the agent keeps track of static and dynamic information about all the servers, selects one of these on the basis of their static (hardware and software) and dynamic (load balancing, effective networks bandwidth) features and notifies the choice to the client. The client is then able to send directly to the server the data of the required task, and the server can use its installed software libraries to perform the computation on client data. Finally the server sends the results back directly to the client and the agent is notified of the completion of the task. For instance the core operation (1) is executed on the servers using the DGEMM routine of LAPACK library, available with the release 2.0 of NetSolve.

3 Computational Experiments

In this section we discuss experiments that we carried out, aimed to evaluate performance behavior of the block matrix multiplication algorithms previously described. For each experiment, the reported total execution times are the average of 10 runs launched at different day hours in order to use the networks with different workloads.

To this aim we used two NetSolve infrastructures. The first one is a system located at our department that we call *LAN system*, and a second one is the system located at the University of Tennessee that we call *WAN system*. For

	LAN system	WAN system
Client location	Dept. Math. and Appl. Univ. Naples - Italy	Dept. Math. and Appl. Univ. Naples - Italy
Agent and servers location	Dept. Math. and Appl. Univ. Naples - Italy	Comp. Science dept. Tennessee Univ.
Servers	7 PCs Pentium III and IV	$>$ 50 PCs and WS
Measured latency and bandwith	$\sim$ 200 μsec $\sim$ 50 Mbits	$\sim$ 140 msec $\sim$ 1.5 Mbits

Fig. 2. features of the NetSolve systems used for our experiments

both systems we report in Figure 2 the main features with special regard to the network. Note the client location is the same in both cases.

A first set of experiments is aimed to verify the effectiveness of the (I,J,K) ordering of the block matrix multiplication algorithm respect to the other versions. With this aim we implemented the two orderings (I,J,K) and (K,I,J) in Figure 1 (namely the best expected version and the worst expected version) on the WAN NetSolve system with square matrix of order n = 250, 500, 1000, 1500, 2000, and a fixed block size $r = 250$, thus NB = 1, 2, 4, 6, 8. In Figure 3 we report the average execution time of the two orderings, where it is evident the better performance of the (I,J,K) ordering of the algorithm. Actually the (I,J,K) versione exploits for every test a smaller average execution time, with a gain of about 50% respect to the (K,I,J) version, because of the smaller syncronization overhead.

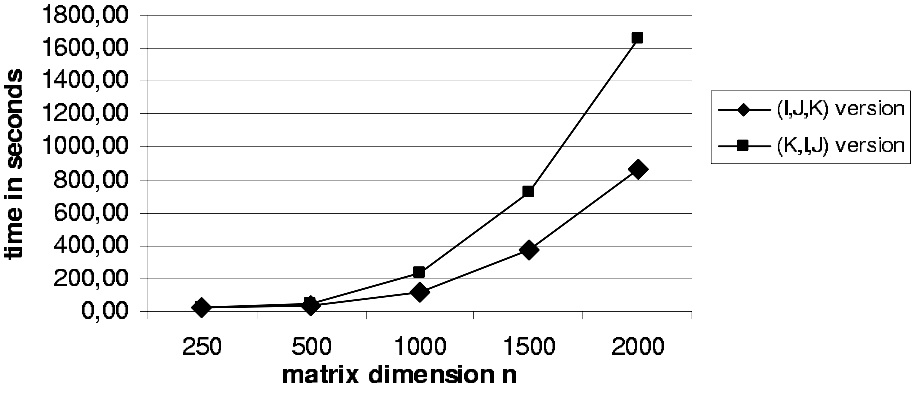

Fig. 3. Average execution time of two orderings of the block matrix multiplication

A second set of experiments is aimed to compare the execution times of the (I,J,K) ordering of the block matrix multiplication algorithm, on the two different NetSolve systems. For these experiments the results are reported in Figure 4. For the same values of n and r used in the previous experiments, we estimate that on the LAN system the total execution time is about 50% smaller with respect to the WAN system, because of the smaller latency and the higher bandwidth of the networks. In order to quantify the performance gain, let us observe the scaled speed up S_2 achieved from $n = 1000$ up to $n = 2000$: on the LAN system $S_2 = 0.23$ whereas on the WAN system $S_2 = 0.14$. These values should be compared with the ideal scaled efficiency $S_2 = 0.5$. Furthermore, we previously stated that, in the ideal case, the execution time grows linearly, but in our experiments we found that the total execution time of the (I,J,K) version grows roughly as NB^2 on the LAN system because of the system overhead. However this asymptotic rate is still smaller than the asymptotic rate of a sequential execution, where the total execution time grows as NB^3. On the other hand, when the (I,J,K) version is executed on the WAN system we found the same grow

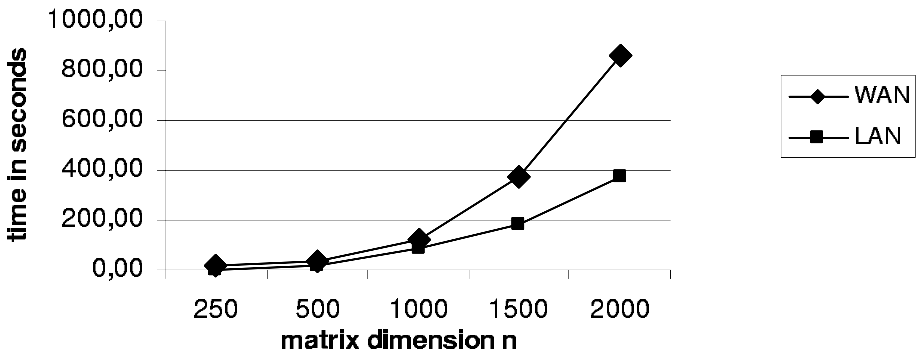

Fig. 4. Average execution time for the (I,J,K) of the matrix multiplication algorithms for the LAN and WAN NetSolve infrastructures

rate NB^3 of a sequential execution, then in this case the proposed approach to the distributed computing is not competitive.

However, in general it is worth to note that efficiency in distributed computing applications is a wrong performance's measure. Actually, the use of a distributed computing infrastructure should not be seen as a high performance computing solution but as a cost / effective solution because it enables the access to computing resources that cannot be conveniently locally located.

4 Conclusions

The development of cost/effective distributed computing environments is enabled by the current advances in networking technologies. The network-based computing environments provides the computational and storage requirements for solving distributed applications. Even thought distributed computing environments can deliver high performances when lot of computations is needed to do embarrassingly parallel tasks, the availability of distributed software infrastructure such as NetSolve and Condor represent a viable and economic solution when other dedicated resources are unavailable. This often requires detailed understanding of the underlying architecture and writing parallel or distributed algorithms needs exploitation of different programming models which are most suitable for addressing communication and synchronization issues. To this aim, we discussed performance behavior of a client/server implementation of some block matrix multiplication algorithms in terms of their synchronization overhead in order to analyze bottlenecks in the algorithms that critically impact performance. While in parallel computing we decompose into parts, in distributed computing we assemble parts [6], and in some cases, composition requires hard synchronization issues.

References

1. D.Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, S. Vadhiyar. User's Guide to NetSolve V. 2.0. Univ. of Tennessee, 2004. See also NetSolve home page - URL: http://icl.cs.utk.edu/netsolve/index.html
2. O. Beaumont, V.Boudet, F. Rastello and Y. Robert. Matrix Multiplication on Heterogene-ous Platforms. in IEEE Trans on Parallel and Distributed Systems, vol. 12 (2001), pp 1033- 1051
3. J. Choi, J. J. Dongarra, and D. W. Walker. PUMMA: Parallel Universal Matrix Multiplication Algorithms. Concurrency: Practice and Experience, Vol. 6 (1994), pages 543-570.
4. J. Dongarra. Performance of Various Computers Using Standard Linear Equations Soft-ware, (Linpack Benchmark Report), University of Tennessee Computer Science Technical Report, CS-89-85, 2005. See also Top500 Supercomputers home page at URL: http://www.top500.org
5. I. Foster and C. Kesselman. Computational Grid. in The Grid: Blueprint for a Future Gen-eration Computing Infrastructure. Foster and Kesselman eds., Morgan Kaufman, 1998.
6. G. Fox. Message Passing from Parallel Computing to the Grid. in IEEE Computing in Sci-ence and Engineering. Sept/Oct 2002
7. GARR home page. - URL: http://www.garr.it
8. M. Litzkow, M.Livny and M.Mutka. Condor: a hunter of idle workstation. In Proc. of 8-th Int. Conf. Distributed Computing Systems, pp. 104-111 IEEE press, 1988. See also URL www.cs.wisc.edu/condor
9. A. Kalinov and A. Lastovetsky. Heterogeneous Distribution of Computations Solving Lin-ear Algebra Problems on Networks of Heterogeneous Computers. Journ. of Parallel and Distributed Computing, Vol. 61 (2001), pp. 520-535
10. Seti@home home page - URL: http://setiathome.ssl.berkeley.edu/