Causes of Blocking Overhead in Message-Passing Programs

Jan Lemeire[1], Erik Dirkx[1]

[1] Parallel Systems lab, Vrije Universiteit Brussel,
Pleinlaan 2, 1000 Brussels, Belgium
{jlemeire, erik}@info.vub.ac.be

Submitted for the 10th Euro PVM/MPI 2003 Conference, Venice, Italy, Sep 29 - Oct 2, 2003

Abstract. This paper studies the idle time of processors during the execution of message-passing parallel programs. Detailed analysis reveals that, besides the well-known load imbalances, blocking overhead can be generated by any part of the execution profile, like communication overhead, message delays or partitioning. We investigated this causal relation and developed an algorithm to determine the causes of blockings. Applied to parallel matrix multiplication, it reveals higher-order p dependencies of the overhead. We argue that for an accurate quantification of the cost of each part of a parallel program, the generated blocking overhead should also be taken into account.

1 Introduction

In literature, processors becoming idle, or blocking overhead, is mainly attributed to load imbalances [1][3][4], but we argue that this is not the only reason for the processors' idle time. Blocking overhead can be caused by any phase of the execution profile, as mentioned in literature [2][6]. Partitioning for example, is in most cases performed sequentially on one processor. This causes at the same time blocking on the other processors, resulting in an important O(p) dependency of the partitioning overhead.

This research fits in our goal to assist the programmer with an automated, understandable and accurate performance evaluation [2][8]. A correct estimate of the cost of each part of the parallel program is therefore indispensable. Thus, the blocking overheads should be added to the cost of the phases that cause the blocking. We developed an approach that starts from the execution profile, measured through code instrumentation, and reveals the causes of all processor's idle time in message-passing programs.

The paper starts with defining the performance metrics for quantifying the impact of the overheads on the performance. In section 3 we define the blocking causal relation and show how it can be attributed. Section 4 explains the concrete implementation, used in section 5 to analyze the overhead of parallel matrix multiplication.

2 Parallel Performance Metrics

We quantify the benefit of parallel processing by the speedup $S=T_{seq}/T_{par}$, how much faster the parallel program runs with respect to the runtime of the sequential version. The portion of T_{par} that is not used for useful computation is therefore considered as lost processor cycles [4], or overhead [1]:

$$Overhead = T_{par} \cdot p - T_s . \tag{1}$$

Hence, the choice of speedup as the performance measure implies that each processor has T_{par} time allocated to perform its part of the job:

$$T_{par} = T^1_{par} = T^2_{par} = \ldots = T^p_{par}. \tag{2}$$

$$T^i_{par} = T^i_{comp} + \sum_j T^{i,j}_{overhead} . \tag{3}$$

With T^i_{comp} the time that processor i performs its part of the useful work and $T^{i,j}_{overhead}$ the time not overlapped with the computation of overhead of type j on processor i. To study the overall impact of the overhead, we can rewrite Eq. (2) as

$$T_{par} = \frac{\sum_i T^i_{par}}{p}. \tag{4}$$

We will develop the overhead equations for processors with equal computing power, which implies that $T_{seq} = \sum_i T^i_{comp}$. Together with Eqs. **(3)** and **(4)**, the speedup can then be rewritten as [6]:

$$S = \frac{p}{1 + \frac{\sum_i \sum_j T^{i,j}_{overhead}}{T_{seq}}}. \tag{5}$$

The impact of the overhead on the speedup is thus reflected by its ratio with the sequential runtime. Let us therefore define this ratio for each overhead type j:

$$Ovh^j = \frac{T^j_{overheads}}{T_{seq}}. \quad (6)$$

These ratios quantify the cost of the overheads. Note that $T^j_{overhead}$ is the totalization of the overhead of type *j* on all processors.

3 Analysis of the Blocking Overhead

Blocking overhead is the processor spent idling during parallel execution, while it has to wait at a synchronization point for information to proceed. Each blocking phase is thus ended by an incoming message of another process. Due to Eq. (1), the initial and the final idle time of each process are also lost cycles and will be included in the blocking overhead.

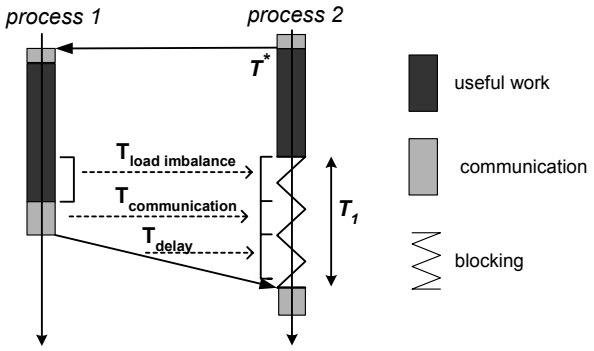

Fig. 1. Detail of an execution profile, showing possible sources of blocking overhead

Whereas blocking overhead is mainly attributed to load imbalances, the execution profile of **Fig. 1** shows that this is an oversimplification. Besides the load imbalance, the blocking T_1 is also a result of communication overhead and the message delay. In general, every computation, communication or network delay can cause blocking. The goal of our research is to determine which phases of the execution profile are responsible for blocking.

3.1 The Causal Relation

A cause-effect relation is recognized by a correlation between two events. A blocking phase happens as a result of a certain phase. This means that without the presence of that phase, the blocking would not occur. Furthermore, blocking of a process is

caused by differences in the execution profile with the process with which it synchronizes after the blocking, as shown in **Fig. 1**. Our algorithm will investigate the times that both processes spent on each phase since their last synchronization (T^* in **Fig. 1**), any difference between their phases indicates a possible cause of blocking.
However, not every imbalance induces an equivalent blocking:
1. A process can simultaneously generate blocking on several other processes (process 2 induces T_2 and T_3 in **Fig. 2a**).

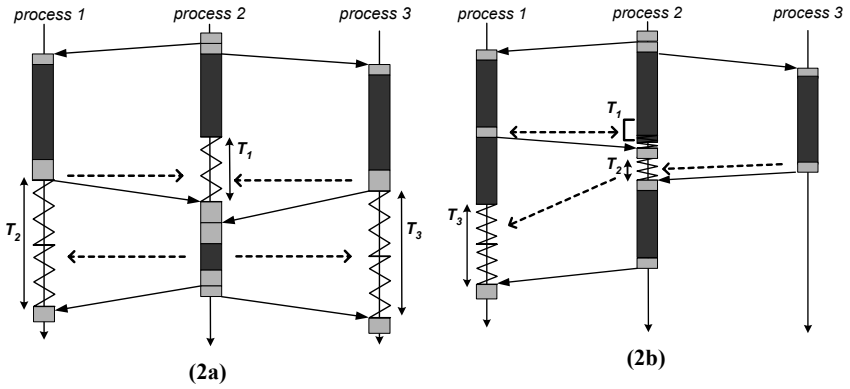

Fig. 2. Example execution profiles of message-passing programs

2. Blocking of one process can be caused by several processes (processes 1 and 3 are both responsible for T_1 in **Fig. 2a**). We solve this case by attributing the blocking overhead proportionally among the contributing processes.
3. Imbalances on different processes can cancel each other out (the load imbalance T_1 on process 2 cancels with the communication overhead on process 1 in **Fig. 2b**). However, in the case of multiple imbalances, the question arises which imbalances should be cancelled. The solution we adopt is to reduce the imbalances proportionally.
4. Imbalances can propagate to other processes, due to the blocking induced on one process that on its turn will cause blocking on another process (T_2 generates blocking T_3 on process 1 in **Fig. 2b**). Therefore, while comparing the execution profiles, blocking phases should be replaced by their cause (T_2 is replaced by the computation imbalance of process 3 for analysis of T_3 in **Fig. 2b**).
For most cases, these pragmatic rules suffice to attribute an unambiguous cause to all the idle times.
Once the cause of each blocking is determined, we can define the blocking generation factor β_j for each phase j:

$$T^j_{blocking} = \beta_j . T^j_{phase} . \quad \textbf{(7)}$$

The factor β_j depends on the imbalances of phase j and on the effect of these phase imbalances.

3.2 Classification of the causes of blocking

Blocking can be caused by all phases of the execution profile. These include the computation, the overheads and the message delays. At the highest level, we identify 4 major classes [2][6][10]:

1. **Communication**, also called interprocess interaction or information movement: the overhead due to the exchange of data between processes, the time that does not overlap with the computation, in the sense of lost processor cycles [4].
2. **Message delays**: the time for the messages to traverse the network, caused by the network latency, the bandwidth and possible network congestion. This delay is not accounted for overhead, only the blocking it causes.
3. **Control of parallelism**: the extra computation that is necessary for management of the parallel processing, like the partitioning or calculations necessary for synchronization.
4. **Workload imbalances**, also called critical path overheads by [2]: the differences among the processes in the computing times T_{comp}^{i} of the useful work.

These major classes can be subdivided further into the different logical parts of the parallel program, so that each part can be studied separately.

3.3 Workload imbalances

The 'traditional' source of blocking overhead, the load imbalances, can be subdivided into 3 distinct parts [2]: the Amdahl blocking, the global load imbalance and the temporal load imbalance.

First, Amdahl's law expresses the limitation of parallelism. There is a serial fraction s of the work that cannot be performed in parallel, during which the other processors will be idling. The generated blocking results in an overhead ratio

$$Ovh_{block}^{amdahl} = (p-1).s \tag{8}$$

Another part of the processor's idle time is due to bad distribution of the workload among the processors. The difference between the *total* workload of the processors is defined as the global load imbalance. However, load imbalances can fluctuate among processes between synchronization points, canceling each other out globally. This does not result in a global load imbalance, but in what we call a temporal load imbalance. Results in parallel discrete event simulation report on low global load imbalances, but considerable temporal load imbalances [7]. The simulation uses a conservative algorithm, consisting of cycles of independent simulation on the different processors that are alternated with barrier synchronizations and intercommunication. Whereas the global load imbalance can be reduced by good partitioning, temporal load imbalances are more difficult to master and can give high slowdowns, especially for increasing p.

4 Implementation

We integrated this analysis into our tool for automated parallel overhead analysis. It uses the same approach as the visual tool XPvm [5], which automatically measures the computation, communication and blocking phases of message-passing parallel programs. Our tool extends this analysis by letting the programmer instrument its code to differentiate the different logical parts of the parallel program, as SCALEA [10]. Starting from the recorded detailed execution profile, we implemented an algorithm that calculates the causes of the blocking phases.

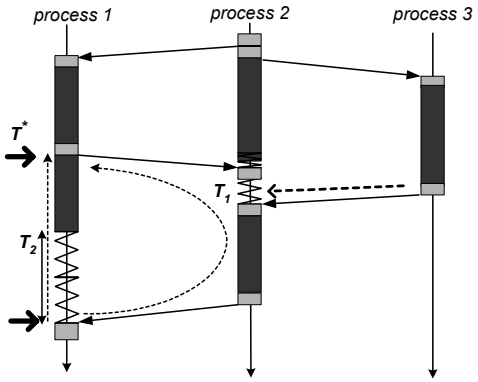

Fig. 3. Algorithm for determining the cause of blocking T_2

The algorithm handles the blocking phases in chronological order. A blocking phase is terminated by an incoming message of another process. The execution profile of this process will be compared with those of the blocking process. This is done by traveling back the execution profile until the previous synchronization of both processes (T^* in **Fig. 3**) or until the beginning of the execution. The duration of all phases are summed and differentiated for each type between both processes. Note that also the message delays are taken into account, so that the paths on both processes are of equal length. Next, all imbalances of the blocking process are cancelled out with imbalances of the other process. The remaining imbalances are considered to be responsible for the blocking phase. Finally, the blocking phase is replaced by all its causes, to be taken into account for the calculation of other blocking (T_1 is replaced by the imbalances of process 3 in **Fig. 3**).

5 Experiments

We illustrate our research with the overhead analysis of parallel multiplication of matrices, $C = A \times B$. The runtime to multiply two dense $n \times n$ matrices is of $O(n^3)$, making it worth for being computed in parallel for high values of n. The parallel algorithm starts with partitioning the matrices in r strips of contiguous rows and c

strips of contiguous columns, where $r \times c = p$ and r close to $\sqrt{p}$. Then, blocks of size $n/r \times n/c$ of matrices A and B are attributed to each processor. The p processes are labeled from $p_{0,0}$ to $p_{r-1,\, c-1}$. Process $p_{i,j}$ will compute submatrix $C_{i,j}$, so it requires all submatrices $A_{i,k}$ and $B_{l,j}$ for $0 \le k < r$ and $0 \le l < c$. To acquire these blocks, an all-to-all broadcast of matrix A's blocks is performed in each row of processes $p_{i,j}$, and an all-to-all broadcast of matrix B's blocks in each column [6][9].

We identify three types of overheads: the partitioning, the communication and the blocking. The load imbalances can be neglected (< 0.2%). The blocking overhead is mainly caused by the communication overhead, the message delays and the partitioning. The experiments are performed for n = 200 on a cluster of eight 333MHz Pentium II processors, connected by a 100Mb/s non-blocking switch.

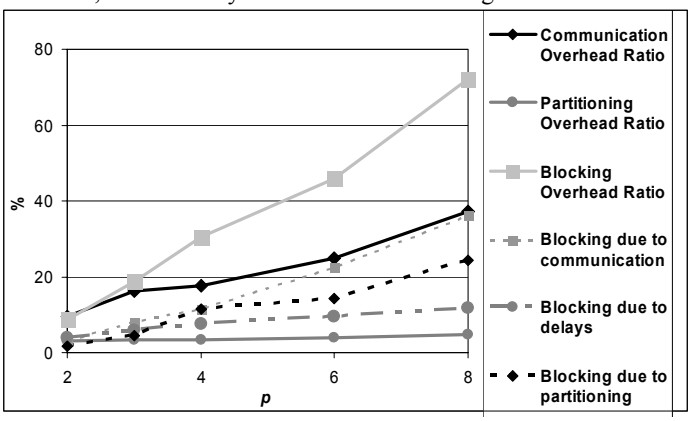

Fig. 4. Overheads of Parallel Matrix Multiplication of 200x200 Matrics

It is shown in **Fig. 4** that the partitioning time stays constant for increasing p and the communication is sublinear of $O(\sqrt{p})$, which reflects the theoretical analysis [6]. The blocking overhead however evolves quite linear for increasing p and increases faster than the communication overhead. To investigate this, table 1 shows the blocking factors β_j.

Table 1. Blocking factors in Matrix Multiplication

	2	3	4	6	8
$\beta_{communication}$	0,32	0,50	0,65	0,90	0,97
β_{delay}	0,41	0,37	0,43	0,38	0,32
$\beta_{partitioning}$	0,57	1,39	3,41	3,64	5,06

It shows that the blocking due to message delays evolves constant with the communication. On the other hand, blockings generated by partitioning and communication increase faster than their cause. This results in a higher-order p dependency. Partitioning and communication overhead will be both of $O(p)$. The linear increase of $\beta_{communication}$ is due to the increasing intercommunication between the processes. Each process has to wait its turn to receive its initial blocks and for the blocks of the row and col-

umn broadcasts. Especially the last process to receive its initial blocks will block the other processes. In this way, the blocking overhead caused by the communication propagates rapidly to all processes and causes an O(p) cost.

7 Conclusion

Indispensable for adequate optimization of parallel programs is an accurate insight in the different overheads. We showed that the blocking caused by each part of the parallel program should be added to its cost. To determine the causes of the idle times, a pragmatic algorithm was developed and added to our tool that measures and analyzes the execution profile of message-passing programs. The dependency of the performance on the number of processors is shown to be non-trivial on non-dedicated clusters.

8 References

1. Bane, M.K. and Riley, G.D.: Automatic Overheads Profiler for OpenMP Codes. In: proceedings of EWOMP2000 conference, Edinburgh, Scotland (2000).
2. Bull, J.M.: A Hierarchical Classification of Overheads in Parallel Programs. In: Proceedings of First IFIP TC10 International Workshop on Software Engineering for Parallel and Distributed Systems, Chapman Hall, (March 1996) 208-219.
3. Clematis, A. and Corana, A.: Modeling performance of heterogeneous parallel computing systems. In: Parallel Computing 25, Elsevier Science (1999).
4. Crovella, M. E. and Leblanc, T.J.: Parallel Performance Prediction using Lost Cycles Analysis. In: Proc. of Supercomputing '94, IEEE Computer Society (1994).
5. Kohl, J.A. and Geist, G.A.: XPVM 1.0 User's Guide. Tech. Rep. 12981, Computer Science and Mathematics Division, Oak Ridge National Laboratory (1995).
6. Kumar, V., Grama, A., Gupta, A. and Karypsis, G.: Introduction to Parallel Computing. Design and Analysis of Algorithms. Benjamin Cummings, California (1994).
7. Lemeire, J. and Dirkx, E.: Performance Factors in Parallel Discrete Event Simulation. In: Proc. of the 15th European Simulation Multiconference, Prague (2001).
8. Pancake, C.M.: Applying Human Factors to the Design of Performance Tools. In: Proc. of the 5th Euro-Par Conf., Springer (1999).
9. Schmidt, B., and Sunderam, V.: Empirical Analysis of Overheads in Cluster Environments. Concurrency: Practice and Experience 6, 1 (February 1994), 1-32.
10. Truong, H-L and Fahringer, T.: Performance Analysis for MPI Applications with SCALEA. In Proc. of the 9th European PVM/MPI Conf., Linz, Austria (September 2002).