Distributed Iterative Solution of Numerical Simulation Problems on Infiniband and Ethernet Clusters via the P2PSAP Self-Adaptive Protocol

Serge Romaric Tembo, The Tung Nguyen, Didier El-Baz
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Université de Toulouse, LAAS, F-31400 Toulouse, France
Email: tembo.mouafo@laas.fr ttnguyen@laas.fr elbaz@laas.fr

***Abstract*—The distributed iterative solution of numerical simulation problems on Infiniband or Ethernet Clusters via the P2PDC environment is studied. The P2PDC decentralized environment is dedicated to task parallel applications. It has been designed for the solution of large scale numerical simulation problems via distributed iterative algorithms. The P2PDC environment is based on the P2PSAP self-adaptive communication protocol. New functionalities of the P2PSAP communication protocol aimed at using Infiniband clusters are presented. A series of computational results is presented and analyzed.**

***Keywords*-distributed computing, cluster computing, Infiniband, micro-protocols, task parallel model, asynchronous algorithms.**

I. Introduction

In order to obtain good efficiency of High Performance Computing (HPC) applications, new transport protocols have to be designed. Existing transport protocols like TCP [1] and UDP [2] were originally designed to provide ordered and reliable transmission to the application or datagram service, respectively and are no longer adapted to real-time and distributed computing applications. We also note that TCP and UDP cannot reconfigure their own structure. Moreover, the message-based transport protocol seems better suited to HPC applications than the classical stream-based communication.

Recently, new transport protocols have been standardized like SCTP [3] and DCCP [4]. Nevertheless, these protocols still do not offer the modularity needed in the context of HPC.

The P2PSAP self-adaptive communication protocol [5] and P2PDC distributed computing environment [6] have originally been designed for peer-to-peer HPC applications. The P2PSAP self-adaptive communication protocol is based on the Cactus framework whereby composite protocols and micro-protocols are combined in order to build a desired communication protocol [7]. In this paper, we concentrate on new functionalities of the P2PSAP communication protocol aimed at using Infiniband clusters. The impact of the underlying network on the distributed solution of numerical simulation problems is also studied. We consider distributed synchronous or asynchronous iterative algorithms. We recall that asynchronous iterations are relative to a very general model whereby computations are carried out in parallel without order nor synchronization (see [8]). They permit one to consider distributed iterative computations where machines go at their own pace according to their intrinsic characteristics and computational load.

The paper is structured as follows. Related work is presented in Section II. Section III is devoted to new features of the P2PSAP protocol aimed at using Infiniband networks. Section IV deals with distributed iterative algorithms for the solution of a numerical simulation problem and displays experimental results. Section V deals with conclusions and future work.

II. Related work

MPICH Madeleine [9] has been designed for testbeds with several types of networks. Nevertheless, the modification of internal transport protocol mechanisms in function of elements of context like iterative schemes of computation and network topology is not allowed with MPICH Madeleine.

P2P-MPI [10] is a framework aimed at developing message-passing programs in large scale distributed networks of computers. P2P-MPI is developed in Java and makes use of Java TCP sockets to implement the MPJ (Message Passing for Java) communication library. P2P-MPI uses a single super-node in order to manage machine registration and discovery that may become a bottleneck. P2P-MPI implements a fault tolerance approach using machine replication that may not be efficient and appropriate to connected problems, since the number of machines involved in the computation will multiply greatly.

The P2PDC decentralized environment was originally designed for peer-to-peer distributed computing. It is particularly dedicated to task parallel applications. The environment P2PDC is intended in particular to scientists who want to solve numerical simulation problems via distributed iterative methods that lead to direct and frequent data exchanges between machines like synchronous or asynchronous iterative methods [8]. The P2PDC environment relies on the P2PSAP self-adaptive communication protocol that implements a reduced set of communication operations (P2Psend, P2Preceive and P2Pwait) in order to facilitate programming [5]. The programmer cares only about the choice of distributed scheme of computation, e.g., synchronous or asynchronous, that he wants to be implemented and does not care about the communication mode between any two machines. The P2PSAP communication protocol chooses dynamically the most appropriate communication mode between any two machines according to

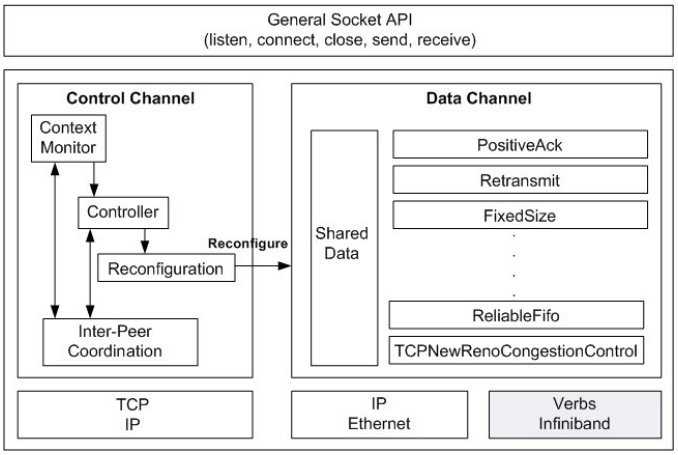

Fig. 1. P2PSAP protocol architecture.

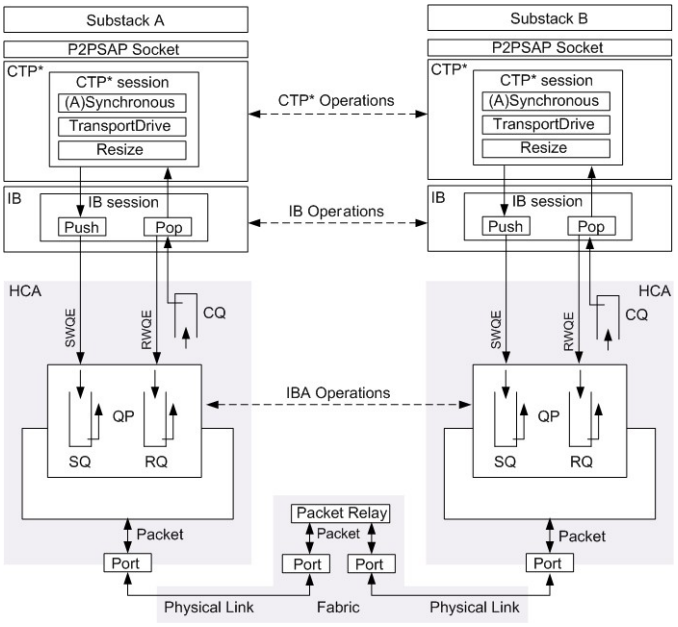

Fig. 2. Data channel communication stack.

decision taken at application level like scheme of computation and elements of context like network topology at transport level. The P2PSAP protocol can also reconfigure itself dynamically to match the very nature of a given algorithm like in the case of the distributed solution of nonstationary problems whereby some synchronizations are needed at every time steps even if an asynchronous scheme of computation has been selected to solve each subproblem. Reference is made to [11] and [6] for studies related to the centralized and decentralized versions of the P2PDC environment, respectively.

III. P2PSAP and Infiniband

The P2PSAP communication protocol has a socket interface and two channels: a control channel and a data channel (see Figure 1). The control channel manages session opening and closure; it captures context information and (re)configures the data channel at opening or operation time; it is also responsible for coordination between machines during reconfiguration process. The data channel is a stack of composite protocols based on the Cactus framework (see [7]). The data channel transfers data packets between machines. The stack of protocols is globaly divided into two layers: a transport layer which is an extension called CTP* of the Configurable Transport Protocol (CTP) [12] and a physical layer that was first designed for IP/Ethernet networks and thanks to the work presented in this paper can encompass also Infiniband networks (see Figure 1). Thus, our contribution concerns the design and insertion of an Infiniband physical layer inside the stack of protocols. We aim also at facilitating self-adaptation of the P2PSAP protocol according to the iterative scheme of computation considered, e.g., synchronous or asynchronous, the location of machines, i.e., in a cluster or in different clusters and the type of network used, e.g., Ethernet or Infiniband. According to the Cactus point of view, the added Infiniband physical layer is a composite protocol that implements communication on the underlying Infiniband network.

We recall that a machine is connected to an Infiniband network via a Host Channel Adapter (HCA) that can have several ports identified by a Local ID (LID) and a Global ID (GID). The transport, network and data link layers are implemented in the silicon of the HCA. The HCA uses internal Direct Memory Access (DMA) engine in order to transfer or retrieve data directly to or from the main memory of a machine. During the application, several communication requests are created that contain information like address of the communication buffer and data size. Each request is transmitted to the appropriate Send Queue (SQ) or Receive Queue (RQ) via the Infiniband Verbs interface. The HCA executes the requests according to the order they are posted and stores the results called Completion Queue Elements (CQE) in the appropriate Completion Queue (CQ).

The insertion of a new layer in the data channel communication stack is not independant of the control channel since it configures and manages the data channel. The control channel detects the type of underlying network at session opening. If the underlying network is Infiniband, then the resources (LID, GID, SQ, RQ, CQ and communication buffers) that are called Queue Pair (QP) context in Infiniband specifications (see [13]) and that are used by the Infiniband composite protocol IB of the data channel are created once for the entire communication duration by the control channel at data channel opening since each consumed Infiniband Verb has a nonnegligible overhead; this is particularly true if the Verb necessitates a kernel transition of the operating system like the memory region registration Verb that is used to allocate memory to communication buffer. The data channel is an Infiniband Reliable Connected (RC) channel in the case of synchronous communication or an Infiniband Unreliable Connected (UC) channel in the case of asynchronous communication since asynchronous iterations can tolerate message loss. We note that the creation of communication resources is not sufficient to establish a RC or an UC Infiniband channel. Each machine must also know some information regarding the remote QP

context. This information includes LID, GID, QP Number (QPN) in addition to initial packet sequence number which is randomly generated by control channel. The exchange of this information is done via the control channel prior to any communication operation of the data channel. As soon as a machine obtains information regarding the remote QP context, the machine will enter the Ready To Receive (RTR) state and then the Ready To Send (RTS) state. During the configuration of data channel, the control channel passes the RTS QP Context to the IB Layer.

The stack of protocols relative to the Infiniband network is as follows: CTP*/IB/Verbs/Infiniband (see Figure 2), where CTP* takes into account Synchronous and Asynchronous micro-protocols. IB is the composite protocol that implements Infiniband communications using resources initially created by the control channel. CTP* contains only basic micro-protocols: TransportDrive that adds port identifiers to each CTP* segment, Resize for segmentation/reassembly and Synchronous or Asynchronous. Functionalities like congestion control, flow control, and error recovery required for synchronous communication are implemented by the HCA in order to reduce CPU protocol overhead. The micro-protocol composition of CTP* depends on the context, i.e., the type of iterative scheme, e.g., synchronous or asynchronous, the location of machines, e.g., intra or inter cluster and the type of underlying network, e.g., Infiniband or Ethernet. Table I shows micro-protocols composition of CTP* in the different contexts.

Unlike in the Ethernet asynchronous context, the Ethernet synchronous context requires reliability in order to ensure that application is not going to be blocked by message loss. This is ensured via SequencedSegment, Retransmit, RTTEstimation, PositiveAck and DuplicateAck micro-protocols. Moreover, this context requires an order delivery property which is ensured via the ReliableFIFO micro-protocol. In order to behave fairly with other flows, congestion control is also required in the case with inter clusters communications whereby we must have reliable flows, this is ensured via Window Congestion Control (denoted by WindCongCtrl) and TCP NewReno Congestion Advoidance (TCPNewRenoCA) micro-protocols. In the Ethernet asynchronous context with inter clusters communications whereby we have unreliable flows, congestion control is ensured via DCCPAck, DCCP Window Congestion Control (DCCPWindCCtrl) and TCP Congestion Avoidance (TCPCongAvoid) micro-protocols.

The composite protocol IB that we have carried out essentially implements two operations of communication:

- the PUSH operation which sends a CTP* segment on the Infiniband network (see Figure 3);

- the POP operation which retrieves a CTP* segment from the network and delivers it to the upper layer (see Figure 4).

Sending a message via Infiniband consists in building and posting a Send Work Request (SWR). The physical memory addresses contained in the Work Queue Element (WQE) must be registered by the consumer. Here, the consumer is the IB composite protocol which performs a mapping between the socket addresses known by CTP* sessions and QP contexts known by HCA. However, waiting for a message to be generated by CTP* before registering a memory region will add a delay to message processing before the transmission over the network. The Resize micro-protocol is introduced in CTP* sessions in order to give a precise value of the maximum size of a CTP* segment to the IB layer, making possible static registration.

TABLE I
TRANSPORT LAYER OF P2PSAP, COMPOSITION OF MICRO-PROTOCOLS,

Context	synchronous				asynchronous			
	intra		inter		intra		inter	
Micro-protocols	IB	Eth	IB	Eth	IB	Eth	IB	Eth
TransportDrive	X	X	X	X	X	X	X	X
Resize	X	X	X	X	X	X	X	X
Synchronous	X	X	X	X				
Asynchronous					X	X	X	X
SequencedSegment		X		X				
PositiveAck		X		X				
Retransmit		X		X				
RTTEstimation		X		X				
ReliableFIFO		X		X				
DuplicateAck				X				
WindCongCtrl				X				
TCPNewRenoCA				X				
DCCPAck								X
DCCPWindCCtrl								X
TCPCongAvoid								X

At data channel establishment, the control channel registers a memory region that serves as communication buffers. This pinned region is divided into two parts: a send buffer and a receive buffer. Figure 3 displays how data are stored in the buffer in order to send a segment. In the sequel, we detail the management of free spaces in the buffer. The send buffer is divided into 1000 blocks with 8128 bytes. Each block can contain only one CTP* segment with 8128 bytes. For each CTP* segment to send, the Infiniband composite protocol IB looks for a free block in the send buffer in order to store the data; the data are then written in the block. The address of the block and the size of the segment are included in the SWR posted to the SQ. A thread continuously polls the CQ. When a Completion Queue Element (CQE) containing the identifier (ID) of a SWR is retrieved from CQ, the block that was used by this SWR can be reused without any risk of inconsistency between two successive writings on this block. A relation between the SWR ID and the block position in the send buffer allows the process which continuously polls the CQ to find out the block and set it free by inserting its position at the end of the list of free blocks.

Note that registration overhead is reduced by reusing blocks of the send buffer. The number of blocks is limited to 1000 in order to guarantee a reasonable memory consumption. A linked list is used to store the position of free blocks in order to accelerate the search for free blocks. The IB layer uses always the block whose position is at the head of the linked list to post a SWR, the head is then moved to the next item; this position is inserted at the end of the list when the SWR Completion is retrieved from the CQ. So, the IB layer always keeps track of the position of free blocks in the send buffer

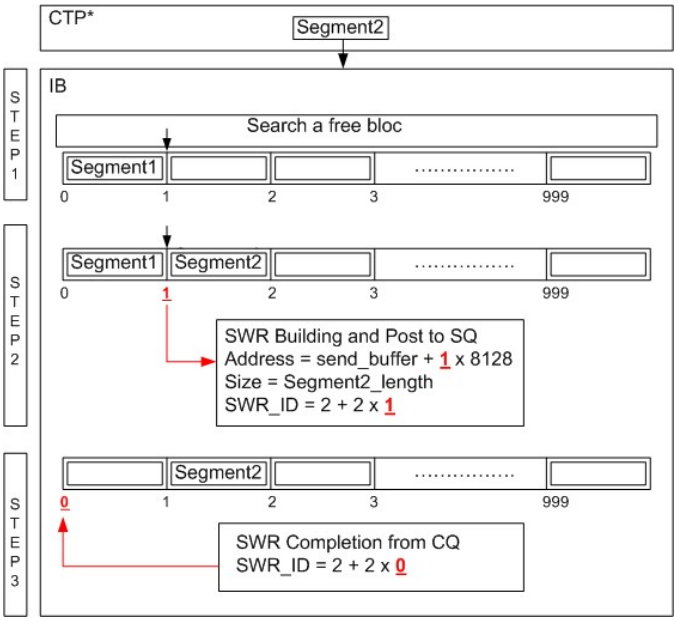

Fig. 3. PUSH operation.

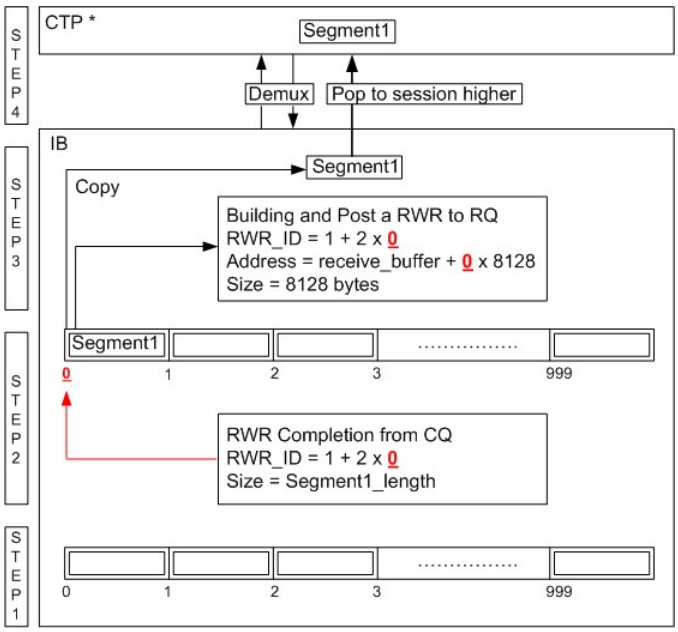

Fig. 4. POP operation.

and does not waste time to find out a free block. Figure 3 illustrates the process:

- the search for a free block (step 1);
- writing the data segment in the free block, building and posting a SWR (step 2);
- retrieving the result of a SWR from CQ and releasing the block used (step 3).

Figure 4 displays how IB retrieves data from the receive buffer. The receive buffer is initially divided into 1000 blocks with 8128 bytes (see step 1). At data channel establishment, the control channel posts 1000 RWR (Receive Work Request) to RQ. Each RWR contains a block address. When the HCA receives the data, it consumes the first WQE in the RQ, writes the data in the block and places a CQE in the CQ. When IB retrieves a CQE, it calculates the position of the block from the ID of Receive WQE (RWQE) consumed (step 2). The data of the segment contained in the block that was found are copied to another memory location. This allows the reuse of the address of this block in another RWR built and posted immediately (step 3) simultaneously with the delivering of the segment to CTP* after a demultiplexing operation required to find out the appropriate session at higher level, i.e., the CTP* session (step 4).

IV. Experimental results

We consider the obstacle problem which occurs in mechanics and finance and can be formulated as follows.

$$\begin{cases} Find\ u^*\ such\ that \\ A.u^* - f \geq 0, u^* \geq \phi\ everywhere\ in\ domain\ \Omega, \\ (A.u^* - f)(\phi - u^*) = 0\ everywhere\ in\ \Omega, \\ B.C., \end{cases}$$

where $\phi \in \mathbb{R}^2 (or\ \mathbb{R}^3)$ is an open set, A is an elliptic operator, ϕ a given function and $B.C.$ are the boundary conditions on $\partial\Omega$. The discretization of the obstacle problem leads to the following fixed point problem.

$$\begin{cases} Find\ u^* \in V\ such\ that \\ u^* = F(u^*), \end{cases} \quad (1)$$

where V is an Hilbert space and the mapping $F : v \to F(v)$ is a fixed point mapping from V into V. We consider the distributed solution of fixed point problem (1) via the projected Richardson method combined with several iterative schemes of computation; reference is made to [11] for the mathematical formulation of synchronous and asynchronous projected Richardson methods. The convergence of asynchronous projected Richardson method has been established in [14]. The choice of scheme of computation has important consequences on the efficiency of the distributed iterative algorithm. The interest of asynchronous iterations for various problems including boundary value problems and optimization has been shown in [8], [15], [16], [17], [18], [19].

TABLE II
Machine characteristics and sequential computational time

Site	Cluster	Processor	Mem.	Time
Nancy	Graphene	Intel Xeon 2.53 GHz	16 Gb	9523 s

We display now and analyze experimental results. We concentrate on distributed synchronous and asynchronous iterative algorithms carried out via P2PDC and P2PSAP for a 3D obstacle problem and parallelepiped-based domain. We consider a cubic domain with 256^3 points. Experiments have been carried out on the Graphene cluster of the Grid'5000 testbed [20]. Machine characteristics and the sequential computational time are given in Table II. Table III displays the computational time of synchronous and asynchronous distributed algorithms on cluster Graphene with up to 64 machines and 1 Gb/s Ethernet

TABLE III
DISTRIBUTED CASE, COMPUTATIONAL TIMES

Machines	synchronous		asynchronous	
	Infiniband	Ethernet	Infiniband	Ethernet
4	2543 s	2862 s	2749 s	2794 s
8	1399 s	1794 s	1410 s	1420 s
16	770 s	1118 s	710 s	733 s
32	437 s	711 s	360 s	376 s
64	247 s	448 s	180 s	192 s

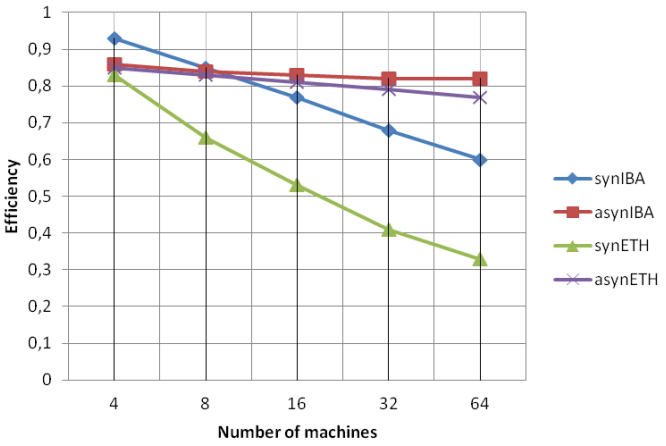

Fig. 5. Efficiency of distributed iterative algorithms.

or 20 Gb/s Infiniband network. Figure 5 shows the efficiency of synchronous and asynchronous distributed algorithms on Infiniband or Ethernet network; the different iterative methods are denoted by synIBA, asynIBA, synETH and asynETH, respectively. Experimental results show that synchronous iterative algorithms carried out on Ethernet network do not scale up well. Their efficiency with 64 machines of the same cluster is close to 30 % (see Figure 5). The new functionalities of P2PSAP aimed at using Infiniband networks permit one to improve the efficiency of synchronous iterative algorithms (the increment is up to 30%). This is mainly due to the reduction of synchronization overhead and fastest communication with Infiniband. Infiniband has low latency and high bandwidth; the Infiniband Adapter has also capability to bypass kernel which off-loads some protocol processing from the CPU; this saves CPU cycles to the benefit of computation. We note also that the combination of asynchronous iterative algorithms with P2PDC is very efficient due to the lack of synchronization overhead and idle time resulting from synchronization. The implementation of these algorithms on Infiniband network permit one to further improve their efficiency.

V. CONCLUSION AND FUTURE WORK

We have proposed extensions to the P2PSAP communication protocol in order to natively use Infiniband network for HPC applications. These new features have permitted us to improve the efficiency of all distributed iterative algorithms that we have carried out, i.e., synchronous and asynchronous algorithms. In particular, we note that the combination of asynchronous iterative algorithms with P2PDC using Infiniband is very efficient: the efficiency is close to 82% with 64 machines.

We are presently working on improvements to the Infiniband functionalities of the P2PSAP protocol in order to accelerate data transmission by bypassing fragmentation/reassembly at CTP* level to finally have only one send request per message instead of one request per CTP* segment. Leaving this functionality to HCA will reduce CPU time wasted in protocol handling. We also plan to deal with heterogeneity. More precisely, we will manage communications between Ethernet clusters and Infiniband clusters and between distants Infiniband clusters connected via an IP/Ethernet network.

ACKNOWLEDGMENT

Part of this study has been made possible with the support of ANR-07-CIS7-011 grant. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed.

REFERENCES

[1] "Transmission Control Protocol (TCP)," in *RFC 793*, 1981.
[2] "User Datagram Protocol (UDP)," in *RFC 768*, 1980.
[3] "Stream Control Transmission Protocol (SCTP)," in *RFC 2690*, 2000.
[4] E. Kohler, M. Handley, and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," in *RFC 2582*, 1999.
[5] D. El Baz and T. T. Nguyen, "A self-adaptive communication protocol with application to high performance peer-to-peer distributed computing," in *Proc. of the 18th Conference on Parallel, Distributed and network-based Processing*, 2010, pp. 323–333.
[6] B. Cornea, J. Bourgeois, T. T. Nguyen, and D. El Baz, "Performance prediction in a decentralized environment for peer-to-peer computing," in *Proceedings of the 25th IEEE Symposium IPDPSW 2011 / HOTP2P 2011*, Anchorage, USA, 2011, pp. 1613—1621.
[7] M. Hiltunen, "The Cactus approach to building configurable middleware services," in *DSMGC2000*, Nuremberg, Germany, 2000.
[8] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc. (republished in 1997 by Athena Scientific), 1989.
[9] O. Aumage and G. Mercier, "MPICH/Madeleine: a true multi-protocol MPI for high performance networks," in *15th International Parallel and Distributed Processing Symposium (IPDPS'01)*, 2001.
[10] S. Genaud and C. Rattanapoka, "A Peer-to-Peer framework for message passing parallel programs," ser. Advances in Parallel Computing, F. Xhafa, Ed. IOS Press, Jun. 2009, vol. 17, pp. 118–147.
[11] T. T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, and M. Chau, "High performance peer-to-peer distributed computing with application to obstacle problem," in *Proceedings of the 24th IEEE Symposium IPDPSW 2010 / HOTP2P*, Atlanta, USA, 2010.
[12] G. Wong, M. Hiltunen, and R. Schlichting, "A configurable and extensible transport protocol," in *Proceedings of IEEE INFOCOM*, 2001.
[13] "Infiniband architecture specification, volume 1, release 1.2.1," 2007.
[14] P. Spiteri and M. Chau, "Parallel asynchronous Richardson method for the solution of obstacle problem," in *Proc. of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, 2002, pp. 133–138.
[15] G. M. Baudet, "Asynchronous iterative methods for multiprocessors," *J. Assoc. Comput. Mach.*, vol. 2, pp. 226—244, 1978.
[16] D. El Baz, "M-functions and parallel asynchronous algorithms," *SIAM Journal on Numerical Analysis*, vol. 27, no. 1, pp. 136–140, 1990.
[17] D. El Baz, "Asynchronous gradient algorithms for a class of convex separable network flow problems," *Computational Optimization and Applications*, vol. 5, pp. 187—205, 1996.
[18] D. El Baz, A. Frommer, and P. Spiteri, "Asynchronous iterations with flexible communication: contracting operators," *Journal of Computational and Appied Mathematics*, vol. 176, pp. 91—103, 2005.
[19] M. Chau, D. El Baz, R. Guivarch, and P. Spiteri, " MPI implementation of parallel sub-domain methods for linear and nonlinear convection-diffusion problems," *Journal of Parallel and Distributed Computing*, vol. 67, pp. 581—591, 2007.
[20] "Grid5000 platform," http://www.grid5000.fr. [Online]. Available: http://www.grid5000.fr