serie ad

WP-AD 2013-10

Assessing Gains from Parallel Computation on Supercomputers

Lilia Maliar

Ivie

Working papers
Working papers
Working papers

La Serie AD es continuadora de la labor iniciada por el Departamento de Fundamentos de Análisis Económico de la Universidad de Alicante en su colección "A DISCUSIÓN" y difunde trabajos de marcado contenido teórico. Esta serie es coordinada por Carmen Herrero.

The AD series, coordinated by Carmen Herrero, is a continuation of the work initiated by the Department of Economic Analysis of the Universidad de Alicante in its collection "A DISCUSIÓN", providing and distributing papers marked by their theoretical content.

Todos los documentos de trabajo están disponibles de forma gratuita en la web del Ivie http://www.ivie.es, así como las instrucciones para los autores que desean publicar en nuestras series.

Working papers can be downloaded free of charge from the Ivie website http://www.ivie.es, as well as the instructions for authors who are interested in publishing in our series.

Versión: diciembre 2013 / Version: December 2013

Edita / Published by:
Instituto Valenciano de Investigaciones Económicas, S.A.
C/ Guardia Civil, 22 esc. 2 1º - 46020 Valencia (Spain)

WP-AD 2013-10

Assessing Gains from Parallel Computation on Supercomputers

Lilia Maliar*

Abstract

We assess gains from parallel computation on Backlight supercomputer. We find that information transfers are expensive. To make parallel computation efficient, a task per core must be sufficiently large, ranging from few seconds to one minute depending on the number of cores employed. For small problems, the shared memory programming (OpenMP) leads to a higher efficiency of parallelization than the distributive memory programming (MPI).

Keywords: Parallel Computation; Information transfers; Speedup; Supercomputers; OpenMP; MPI; Blacklight.

JEL classification numbers: C63, C68.

---

* Hoover Institution, Stanford University and University of Alicante. Correspondence to: Lilia Maliar, T24, Hoover Institution, 434 Galvez Mall, Stanford University, Stanford, CA 94305-6010, USA; Tel. 650-725-3416; e-mail: maliarl@stanford.edu. I acknowledge support from the Hoover Institution and Department of Economics at Stanford University, University of Alicante, Ivie, MCI and FEDER funds under the projects SEJ-2007-62656 and ECO2012-36719. I thank Eric Aldrich, Yongyang Cai, Kenneth L. Judd, Serguei Maliar and Rafael Valero for useful discussions of several issues related to parallel computation. I acknowledge XSEDE grant TG-ASC120048, and I thank Phillip Blood and Rick Costa, scientific specialists from the Pittsburgh Supercomputing Center, for excellent technical support.

1 Introduction

The speed of processors was steadily growing over the last few decades. However, this growth has a natural limit (because the speed of electricity along the conducting material is limited and because a thickness and length of the conducting material is limited). The recent progress in solving computationally intense problems is related to parallel computation. A large number of central processing units (CPUs) or graphics processing units (GPUs) are connected with a network and are coordinated to perform a single job. Each processor alone does not have much power but taken together, they can form a supercomputer.

An important role in parallel computation plays communication. A zero core (master) splits a large problem into many small tasks and assigns the tasks to other cores (workers); workers complete their tasks and return their results to the master; and the master aggregates the results and produces the final output. In a typical application, the master and workers exchange information during the process of computation. If the information exchange is not sufficiently fast, the gains from parallelization may be low or absent. Instead of parallel speedup, we may end up with parallel slowdown.

In this paper, we assess the speed of communication on Blacklight supercomputer – a state-of-art high performance computing machine with 4096 cores. Our numerical examples are implemented using C programming language and two alternative parallelization methods: shared memory programming (OpenMP) and distributed memory programming (MPI). For high performance computing such as Blacklight, the network is designed and optimized for fast communication between the cores using a three-dimensional torus topology design. Still, the information exchange between the cores on Blacklight is far slower than that between the cores on conventional desktops and laptops.

We assess gains from parallel computation on Backlight supercomputer, and we find that information transfers are expensive. Our experiments show that the problem must be sufficiently large to insure gains from parallelization on supercomputers. The optimal size of the task assigned to each core must be at least few seconds if several cores are used, and it must be a minute or more if a large number (thousands) of cores are used. We also find that for small problems, OpenMP leads to a higher efficiency of parallelization than MPI.

A key novel feature of our analysis is that it focuses on state-of-art supercomputers. High performance computing is commonly used in many fields (physics, biology, engineering, etc.) but applications to economics are scarce. Usefulness of high performance computing for economic applications was shown in pioneering work of Amman (1986, 1990), Nagurney and Zhang (1998); and also, see Nagurney (1996) for a survey. This earlier literature focuses on mainframes and is closest to our analysis. More recent literature on parallel computation rely on desktops and clusters of computers. Doornik et al. (2006) review many applications of parallel computation in econometrics; Creel (2005, 2008) and Creel and Goffe (2008) illustrate the advantages of parallel computation in the context of several economically relevant examples; Sims et al. (2008) employ parallel computation in the context of large-scale Markov switching models; Aldrich et al. (2011), and Morozov and Mathur (2012) apply GPU computation to solve dynamic economic models; Durham and Geweke (2012) use GPUs to produce sequential posterior simulators for applied Bayesian inference; Cai et al. (2012) apply high throughput computing (Condor network) to implement value function iteration; Villemot (2012) use parallel computation in the context of sovereign debt models; and finally, Valero et al. (2013) review parallel computing tools available in MATLAB and illustrate their application in the context of the Smolyak methods for solving large-scale dynamic economic models.

The rest of the paper is organized as follows. In Section 2, we describe the parallel computation paradigm and its applications in economics. In Section 3, we discuss state-of-art supercomputers and assess their speed in communicating information. Finally, in Chapter 4, we conclude.

2 Parallel computation

In this section, we describe the advantages and limitations of parallel computation and show examples of parallelizable problems in economics.

2.1 Why do we need parallel computation?

In the past decades, the speed of computers was steadily growing. Moore (1965) made an observation that the number of transistors on integrated circuits doubles approximately every two years, and the speed of processors doubles approximately every 18 months (both because the number of transistors increases and because transistors become faster). The Moore law continues to hold meaning that in 10 years, computers will become about 100 times faster.

What shall we do if a 100-times speedup is not sufficient for our purposes or if a 10-year horizon is too long for us to wait? There is an alternative source of computational power that is available at present – parallel computation: we split a large problem into smaller subproblems, allocate the subproblems among multiples workers (processors), and solve all the subproblems at once. Serial desktop computers have several central processing units (CPUs) and may have hundreds of graphics processing units (GPUs), and a considerable reduction in computational expense may be possible. Supercomputers have many more cores (hundreds of thousands) and have graphical cards with a huge number of GPUs. Each processor in a supercomputer is not (far) more powerful than a processor on our desktop but pooling their efforts gives them a high computational power. Executing 10,000 tasks in parallel can increase the speed of our computation up to a factor of 10,000. This is what supercomputers are.

2.2 Parallelizable problems in economics

Many applications in economics can benefit from parallel computation. The easiest case for parallelization are jobs that are composed of a large number of independent tasks. This case is known in computer science literature as *naturally parallelizable jobs*.

One example of naturally parallelizable jobs is stepwise regressions in econometrics: we run a large number of regressions of a dependent variable on different combinations of independent variables to see which combinations produce best results, and we may run each regression on a separate core. Another example is sensitivity analysis: we solve an economic model under many different parameterizations either because we want to study how the properties of the solution depend on a specific parameterization or because we want to produce multiple data sets for estimating the model's parameters (nested fixed point estimations). In this case, we may solve a model for each different parameter vector on a separate core. See Doornik et al. (2006) for a review of applications of parallel computation in econometrics. Other examples of naturally parallelizable jobs are matrix multiplication, exhaustive search over a discrete set of elements, optimization of a function over a region of state space, etc.

However, most applications in economics cannot be parallelized entirely. A typical application will contain some parts that are naturally parallelizable, other parts that cannot be parallelized and must be executed serially and other parts that can be parallelized but require information exchange between cores in the process of computation. For example, in numerical methods for solving economic models, one can paralelize expensive loops across grid points and or integration nodes. Since such loops appear inside an iterative cycle, after each iteration, a master core needs to gather the output produced by all workers and to combine it in order to produce an input for the next iteration; iterations continue until convergence is achieved. In this case, the computer code is a sequence of alternating parallel and serial computations; see, e.g., Creel (2005, 2008), Creel and Goffe (2008), Aldrich et al. (2011), Morozov and Mathur, Cai et al. (2012), Villemot (2012), Valero et al. (2013).

2.3 Limitations of parallel computation

Parallel computation is a promising tool for many problems in economics but it is not automatically useful in every possible context. The limitations of the parallel computation approach are as follows.

i) Not every problem can be parallelized. For example, suppose we need to produce time series for an AR(1) process $x_{t+1} = \rho x_t + \varepsilon_{t+1}$, where $\rho \in (-1, 1)$ and ε_{t+1} is a random variable drawn from a given distribution. To produce each subsequent value x_{t+1} we need to know the previous value x_t and thus, the loop must be executed in a serial manner.

ii) Gains from parallelization are limited by the fraction of code that cannot be parallelized (i.e., that needs to be executed serially), which is referred to in the literature as Amdahl's (1967) law. Indeed, if 50% of time is spent on running nonparallelizable code, we can reduce the total running time by a factor of 2 at most, no matter how many cores we employ.

iii) Different tasks executed in a parallel manner may differ in the amount of time necessary for their execution. For example, when searching for a maximum of a function over different regions of state space, a numerical solver may need considerably more time for finding a maximum in some regions than in others. The most expensive region will determine the speedup and efficiency of parallelization since all the workers will have to wait until the slowest worker catches up with the rest.

iv) The cost of information transfers between multiple cores may be high and may dramatically reduce the gains from parallelization. Instead of a parallel speedup, we may have a parallel slowdown.

3 Supercomputers

For desktops, the information exchange between CPU is very fast. The information exchange is slower for GPU computations but is still very fast; see Aldrich (2010), Morozov and Mathur, and Valero et al. (2013). For supercomputers, the information exchange is far slower and may reduce dramatically gains from parallelization even in applications that are naturally suitable for parallelization. The goal of this section is therefore to determine how large a task per core should be to obtain sufficiently high gains from parallelization on supercomputers. we first discuss the capacities of modern supercomputers, and we then assess the cost of information transfers on Blacklight supercomputer.

3.1 Supercomputer types

Three different types of supercomputers are distinguished in computer science literature; see Blood (2011).

1. High-performance computing (HPC) runs one large application across multiple cores (either CPUs or GPUs). The user is assigned a fixed number of processors for a fixed amount of time, and this time is over if not used.

2. High-throughput computing (HTC) runs many small applications at once. The HTC computation is opportunistic: the user gets a certain number of cores that nobody uses at that time, and this computer time would be wasted otherwise.

3. Data intensive computing focuses on input-output operations, where data manipulation dominates computation.

We are interested in the first two types of supercomputers, HPC and HTC. Blacklight is an example of an HPC machine. It consists of 256 nodes each of which holds 16 cores, 4096 cores in total. Each core has a clock rate of 2.27 GHz and 8 Gbytes of memory. The total floating point capability of the machine is 37 Tflops, and the total memory capacity of the machine is 32 Tbytes. Blacklight has many software packages installed including C, C++, Fortarn, R, Python, MATLAB, etc. For a detailed description of Blacklight supercomputer, see *http://www.psc.edu/index.php/computing-resources/blacklight*.

Condor is an example of an HTC machine. It is composed of a large net of computers. Computers in the net belong to priority users and are not always free (our own computers can become a part of the Condor network if we give them a permission). Condor software detects computers that are not currently occupied by priority users and assigns tasks to them. It passes messages between masters and workers, queue the tasks, detects failures and interruptions, collects the output and delivers it to users. The Condor network is slower than that of HPC machines but the speed of communication is not essential for many applications, in particular, for those that are naturally parallelizable. For a detailed description of Condor network, see *https://www.xsede.org/purdue-condor*.

High computational power becomes increasingly accessible to economic researchers. In particular, eXtreme Science and Engineering Discovery Environment (XSEDE) portal financed by NSF provides access to supercomputers for US academic/nonprofit institutions. Currently, XSEDE is composed of 17 service providers around the world, see *https://portal.xsede.org*. Blacklight and Condor are two examples of supercomputers that belong to the XSEDE portal. Computer time can be also bought in internet at relatively low prices. For example, Amazon Elastic

Compute Cloud Amazon provides the possibility to pay for compute capacity by the hour; see, e.g., *http://aws.amazon.com/ec2/#pricing.*

3.2 Shared versus distributed memory programming

Using supercomputers requires certain knowledge of the computer architecture and the operational system (typically, Unix), as well as software that distributes and exchanges information between different cores. This is because in addition to our main code, we must design software that splits a given job into smaller jobs, that exchanges information between the different cores in the process of computation and that gathers the information to produce final output.

An important issue for parallel computation is how to share the memory. Two main alternatives are shared memory programming and distributed memory programming.

- *Shared memory programming.* There is a global memory which is accessible by all processors, although processors may also have their local memory. For example, OpenMP software splits loops between multiple threads and shares information through common variables in memory; see *http://www.openmp.org.*
- *Distributed memory programming.* Processors possess their own memory and must send messages to each other in order to retrieve information from memories of other processors. MPI is a commonly used software for passing messages between the processors; see *http://www.mpi-forum.org.*

The advantage of shared memory is that it is easier to work with and it can be used to parallelize already existing serial codes. The drawbacks are that the possibilities of parallelization are limited and that sharing memory between threads can be perilous. The advantage of distributed memory is that it can work with a very large number of cores and is ubiquitous but it is also more difficult to program. There are also hybrids that use distributed memory programming for a coarse parallelization and that use shared memory programming for a fine parallelization.

3.3 Cost of information transfers on Blacklight supercomputer

To assess the cost of information transfers, we implement a simple numerical example on Blacklight supercomputer. We specifically consider a function with a unique input x, which is randomly drawn from a uniform distribution $[0,1]$

$$y = \sin(3x) + \cos(\pi x) + \frac{x^5}{5} + \sqrt{x}\arccos(x) + 8x\exp(x). \tag{1}$$

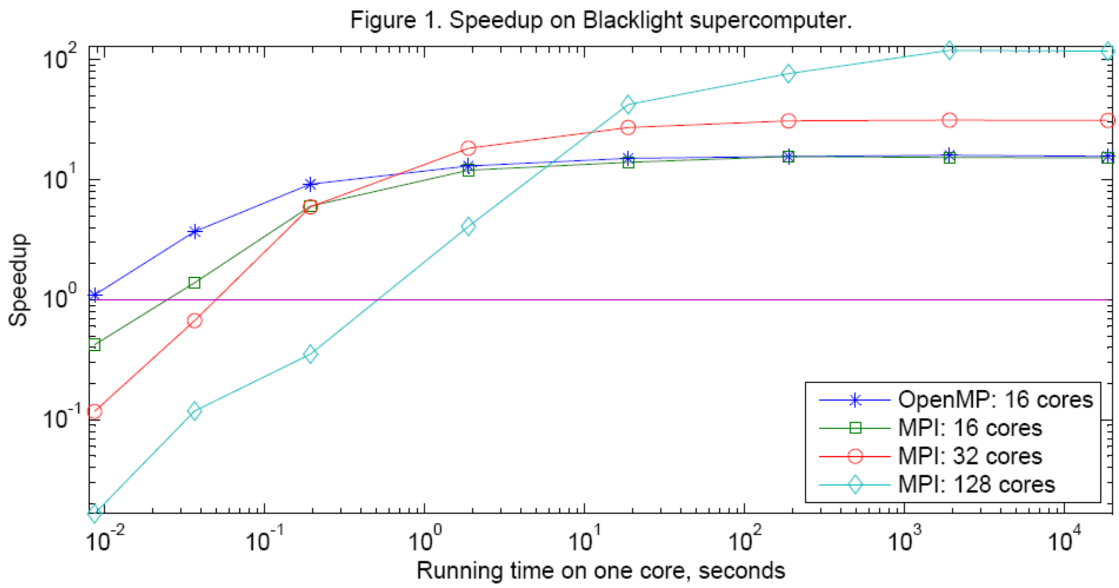
Our objective is approximate the expectation of y using a conventional Monte Carlo integration method $E(y) \approx \frac{1}{n}\sum_{i=1}^{n} y_i$. We study how the computational expense depend on the size of the problem n. We split the problem across multiple cores so that all cores performs tasks of the same size. For example, if the number of cores is 16, each core processes $n/16$ observations.

Our code is written in C programming language (compiled with mpicc). We implement parallel computation using both shared memory programming under OpenMP (with multiple threads) and distributed memory programming under MPI (with point-to-point communication). In the experiments with OpenMP, we parallelize computation across 16 cores of a single Blacklight blade. In the experiments with MPI, we used 16, 32 and 128 cores corresponding to 1, 2 and 4 blades, respectively. In the MPI code, each core (process) runs a copy of the executable (single program, multiple data), takes the portion of the work according to its rank and works independently of the other cores, except when communicating.

In Figure 1, we plot the speedup, which is defined as defined as a ratio

$$\mathcal{S}(N) = \tau_1/\tau_N, \tag{2}$$

where τ_1 and τ_N are the times for executing a job on one core and N cores, respectively.



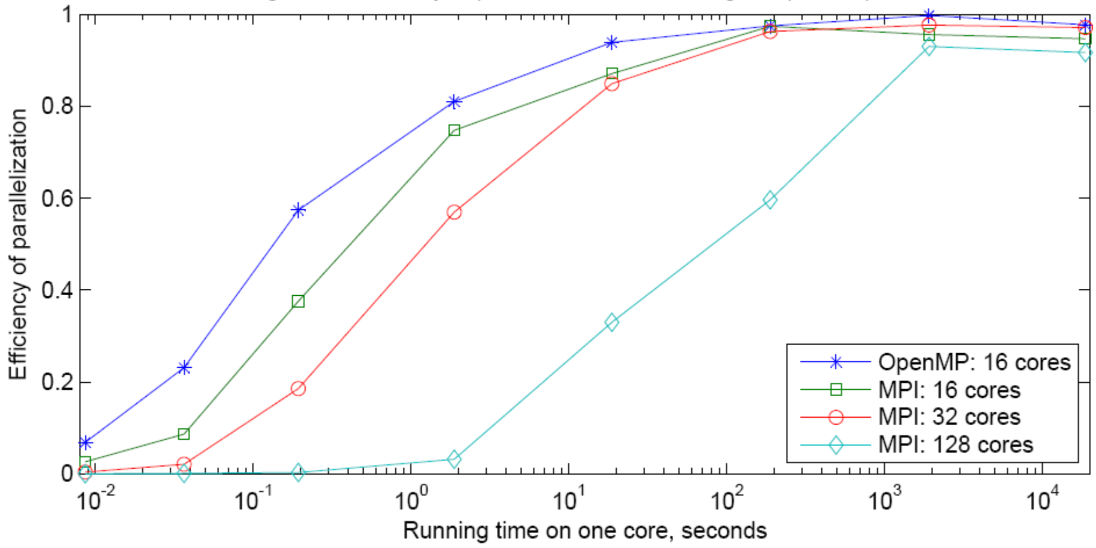
In the experiments using OpenMP, the cost of information transfers is relatively low, in particular because all 16 cores belong to the same blade. Here, we observe positive speedups even for problems of very small sizes. In contrast, in the experiments using MPI, the costs of information transfers dominates gains from parallelization for small problems, and the serial code on just one core runs faster than the parallelized code. We have parallel slowdown instead of parallel speedup. The intersections of the speedup curves with a straight line equal to one indicate the points at which the speedup becomes positive: it ranges from 0.03 seconds for 16-core case to about 0.9 seconds for 128-core case, which correspond to the running time per core ranging from 0.003 to 0.007 seconds respectively. When the problem increases, so do the speedups, approaching the number of cores used in computation.

In Figure 2, we plot the efficiency of parallelization, which is defined as

$$\mathcal{E}\left(N\right)=\frac{\tau_1/\tau_N}{N}. \tag{3}$$

The efficiency shows gains from parallelization τ_1/τ_N relative to the number of cores used N.

Figure 2. Efficiency of parallelization on Blacklight supercomputer.



The tendencies in Figure 2 are parallel to those observed in Figure 1. When the problem is small, the efficiency of parallelization is low, however, when the problem increases, so does the efficiency, approaching one gradually. In the experiments with 16 cores, the efficiency of parallelization using OpenMP is higher than using MPI. Furthermore, our experiements with MPI show that the efficiency of parallelization also depends on the number of cores used: with 16 cores, the efficiency of paralelization of 90% is reached for 20-second problem (2.5 seconds per core), while with 128 cores, a comparable efficiency of parallelization is reached only for 2000-second problem (15.6 seconds per core). Thus, the cost of information transfers increases with the number of cores used. Our sensitivity experiments (not reported) had shown that for larger number of cores, the size of the task per core must be as a minute or even more to achieve high efficiency of parallelization under MPI.

4 Concluding comments

Parallel computing opens a new dimension in numerical analysis in economic. Substantial gains from parallelization are possible even on desktop computers with few cores. Supercomputers have thousands and thousands of CPUs and GPUs that can be coordinated for computationally intensive tasks. Also, they have large memories to record the results. These new possibilities may help to bring economic research to a qualitatively new level in terms of generality, empirical relevance, and rigor of results.

However, to take advantage of this novel technology, we must formulate problems and design codes in a manner which is suitable for parallelization. Furthermore, an important factor to take into account is the cost of information transfers between the cores. The tasks assigned to each core must be sufficiently large to have non-trivial gains from parallelization.

Our numerical assessment shows that the cost of communication is high for supercomputers and may reduce the gains from parallelization dramatically. This is true even for high performance machines, such as Blacklight, whose network is optimized for a fast connection between the cores. We find that the task assigned to a core must be between few seconds and one minute depending on the number of cores used. The efficiency of parallelization is higher under OpenMP than under MPI but the possibilities of parallelization are more limited. Of particular interest appear to be hybrids that use distributed memory programming for coarse parallelization (across blades) and that use shared memory programming for fine parallelization (across cores withing the blade).

Our numerical findings may be useful to researchers who design parallelization codes for supercomputers. The Monte Carlo analysis is a specific example that maybe of limited interest. However, the Monte Carlo code can be readily replaced in our OpenMP and MPI programs by any other code the readers maybe interested in (regression, numerical analysis of equilibrium in some model, etc.). The results of our numerical assessment are suggestive for other applications as well.

References

[1] Aldrich, E. M., Fernández-Villaverde, J., Gallant, R., and J. Rubio-Ramírez, (2011). Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors. Journal of Economic Dynamics and Control, Elsevier 35(3), 386-393.

[2] Amdahl, G. (1967). The validity of single processor approach to achieving large scale computing capabilities". AFIPS procedings, pp. 483-485.

[3] Amman, H., (1986). Are supercomputers useful for optimal control experiments? Journal of Economic Dynamics and Control 10, 127-130.

[4] Amman, H., (1990). Implementing stochastic control software on supercomputing machines. Journal of Economic Dynamics and Control 14, 265-279.

[5] Blood, P. (2011). Getting started using national computing resources. http://staff.psc.edu/blood/ICE11/XSEDE_ICE_July2011.pdf.

[6] Cai, Y., Judd, K. L., Train, G. and S. Wright, (2013). Solving dynamic programming problems on a computational grid. NBER working paper 18714.

[7] Creel, M., (2005). User-friendly parallel computations with econometric examples. Computational Economics 26(2), 107-128.

[8] Creel, M. (2008). Using parallelization to solve a macroeconomic model: a parallel parameterized expectations algorithm. Computational Economics 32, 343-352.

[9] Creel, M. and W. Goffe, (2008). Multi-core CPUs, clusters, and grid computing: a tutorial. Computational Economics 32(4), 353-382.

[10] Doornik, J., N. Shephard and D. Hendry, (2006). Parallel computation in econometrics: a simplified approach. In E. J. Kontoghiorghes (Eds.) Handbook of Parallel Computing and Statistics. Statistics: a series of TEXTBOOKS and MONOGRAPHS, chap. 15, pp. 449-476.

[11] Gallant R.A. (2012). Parallelization strategies: hardware and software (two decades of personal experience). http://www.duke.edu/~arg.

[12] Durham, G. and J. Geweke, (2012). Adaptive sequential posterior simulators for massively parallel computing environments, manuscript.

[13] Moore, G. E. (1965). Cramming more components onto integrated circuits. Electronics 38(8), 1965.

[14] Morozov, S. and S. Mathur, (2012). Massively parallel computation using graphics processors with application to optimal experimentation in dynamic control. Computational Economics 40, 151-182.

[15] Nagurney, A., (1996). Parallel computation, in: H. M. Amman, D. A. Kendrick and J. Rust (Eds.), Handbook of Computational Economics, Vol. 1, Amsterdam: Elsevier, pp. 336-401.

[16] Nagurney, A. and D. Zhang, (1998). A massively parallel implementation of discrete-time algorithm for the computation of dynamic elastic demand and traffic problems modeled as projected dynamical systems. Journal of Economic Dynamics and Control 22(8-9), 1467-1485.

[17] Valero, R., Maliar, L. and S. Maliar, (2013). Parallel speedup or parallel slowdown: is parallel computation useful for solving large-scale dynamic economic models? Manuscript.

[18] Villemot, S., (2012). Accelerating the resolution of sovereign debt models using an endogenous grid method. Dynare working paper 17, http://www.dynare.org/wp.

Ivie

Guardia Civil, 22 - Esc. 2, 1º
46020 Valencia - Spain
Phone: +34 963 190 050
Fax: +34 963 190 055

Website: www.ivie.es
E-mail: publicaciones@ivie.es