DISTRIBUTED MEMORY COMPUTING IN ECONOMICS USING MPI

JEREMY BEJARANO

Abstract. In this project I have two goals: (1) demonstrate the power and scalability of the distributed memory multiprocessing in solving problems in economics and (2) demonstrated how the distributed memory model requires a modified approach to algorithm design. In particular, I will show how to implement a parallel numerical solution to a simple continuous-time optimal control problem (in this case a life-cycle boundary value problem) and will demonstrate resulting speedup. In solving the resulting system of linear equations, I will demonstrate the process of parallelizing the Jacobi method and show how the Jacobi method, in some regards, is preferred over Gauss-Siedel and successive over-relaxation under the distributed memory paradigm. These are done using the Message Passing Interface (MPI)—the de facto standard for parallel computing and features one of it's most noteworthy software libraries, PETSc (Portable, Extensible Toolkit for Scientific Computation).

1. Introduction

Over the past few years, as the development of ever faster monolithic processors has slowed, parallel computing has become more and more visibly important [1]. This is due to the fact that in many situations, the need for greater computational power is the most immediately noticeable constraint holding researchers back from a solution. Seemingly, High Performance Computing (HPC) offers the most readily available solution. But, due to the nature of modern HPC architectures, the software requires special parallel design. Accordingly, parallel computing has taken a central role in various efforts such as climate modeling, protein folding, drug discovery, energy research, and many data analysis application. Considering these trends, it's natural to ask how parallel computing has found use in the study of economics and to ask how it could potentially be used in the future.

1.1. Previous Application in Economics.

Keeping in mind that HPC is still developing and its use in many fields is still very young, it's interesting to see how it has found use in economics. Most application have, naturally, been in econometrics. Optimization routines with many degrees of freedom are a natural application. While many applications of parallel computing are targeted towards general statistics, a few have economics and econometrics specifically in mind. These include, for example, an application of MPI (the message passing interface—the de facto standard for distributed memory computing) to maximum likelihood estimation [12], "parallel strategies for computing the orthogonal factorizations used in the estimation of econometric models" [11], solving finite mixture models [6] [7], even a book chapter on parallel computing in econometrics found in the *Handbook of Parallel Computing and Statistics* [5], and other examples [4]. Another important area of focus is in the numerical solution of economic models—the most popular of which are dynamic stochastic general equilibrium models (DSGE). Some recent work in this area includes a paper on

Date: April 20, 2013.

Key words and phrases. Numerical Methods, Economics, Parallel Computing, MPI, GPU.

sequential Monte Carlo sampling for DSGE models [8], dynamic programming using graphic processing units (GPU) [3], and a parallel plugin for the DSGE solving platform Dynare [2].

1.2. **Possible Future Applications.** Given the existing literature, HPC is yet to impact the field in a major way. New technologies, such as "big data" analytics (like Hadoop), GPUs, and the nearly ubiquitous access to large compute clusters (through "cloud" services or other ways) present clear opportunities for researchers. The challenge remains to find an important application that needs the extra computing power. As best as we can predict, these will likely be applications of big-data, large scale econometrics, complex DSGE models, or agent based simulations. In this paper, I will discuss HPC in general, but focus on distributed memory computing. I will discuss MPI and showcase PETSc (Portable, Extensible Toolkit for Scientific Computation), a well-known and well-developed MPI software library.

2. MPI: The Message Passing Interface

TODO: This section will be filled in later. This will introduce MPI and HPC in general. It will describe the difference between platforms and paradigms and then focus on the role of MPI.

- Threading: OpenMP and PThreads
- MPI
- GPUs: CUDA and OpenCL
- Hadoop

3. Example: Life-Cycle Model of Consumption and Labor Supply

TODO: This will be cleaned up later.

Here are some first results from parallelizing a numerical solution to the optimal control problem taken from Judd (1998) [9]. The solution characterized by the Hamiltonian-Jacobi-Bellman equations is a boundary value problem consisting of a system of linear, first-order differential equations. To solve this BVP, I employ the method of finite differences. I use second order accurate centered differences (stable for these equations) to set up a system of linear equations. I then provide time the execution of the numerical solution in Matlab and then in PETSc using several different numbers of processors.

3.1. **Problem Statement.** A simple case of a life-cycle model is given as follows:

$$\max_c \int_0^T e^{-1\rho t} u(c)\,\mathrm{d}t$$

such that

$$\dot{A} = f(a) + w(t) - c(t)$$
$$A(0) = A(T) = 0$$

where u(c) is a concave utility function over consumption c, $w(t)$ is the wage rate at time t, $A(t)$ is assets at time t, and $f(A)$ is the return on invested assets. The boundary conditions reflect the assumption that assets are initially and terminally zero.

The solution to this problem is derived from conditions on the Hamiltonian (for further details, see Kamien and Schwatrz, 2012 [10]), $H = u(c)\lambda(f(A) + w(t) - c)$. The costate equation is $\dot{\lambda} = \rho\lambda - \lambda f'(A)$. Furthermore, the maximum principle implies the condition

$0 = u'(c) - \lambda$, and thus $c = C(\lambda)$. And this results in the system that characterizing the solution:

$$\dot{A} = f(A) + w - C(\lambda)$$
$$\dot{\lambda} = \lambda(\rho - f'(A))$$

with the same boundary conditions

$$A(0) = A(T) = 0.$$

To simplify the problem, we use the follwing parameters and functional forms.

discount rate	$\rho = 0.05$
time	$t \in [0, 50]$
return of assets	$f(A) = rA(t)$
interest rate	$r = 0.06$
wage function	$w(t) = 0.5 + \frac{5t}{T} - 4\left(\frac{t}{T}\right)^2$
utility function	$u(c) = ln(c)$

After solving for these equations, we should end up with the following results, plotted in Figure 1.

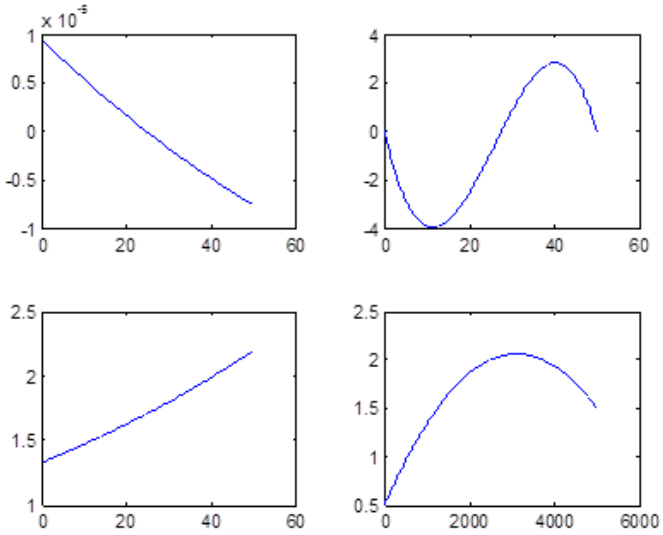


FIGURE 1. Solution values plotted v. time: top-Left, RHS; top-right, assets; bottom-left, consumption; bottom-right, wages

You will notice that these solutions, as plotted in Figure 1, match our intuition. Given the expected structure of lifetime wages, the agent will store up assets so as to smooth consumption over time. Consumption, however, is increasing over time. This is to account for the time discounting. Given discouting, consumption is fully smoothed over lifetime.

3.2. **Speedup Measurements.** In the numerical solution of this BVP, I employ the method of finite differences. I use second order accurate centered differences (stable for these equations) to set up a system of linear equations. To solve this system of linear equation I use a fairly well-known parallel library for scientific computing, PETSc (Portable, Extensible Toolkit for Scientific Computation). For comparison, I solve the same problem in Matlab. All timings are performed on the BigMem nodes of BYUs Fulton Supercomputing Lab. The BigMem machines have 4 Quad-core AMD Opteron 8356 processors and 128 GB memory per node.

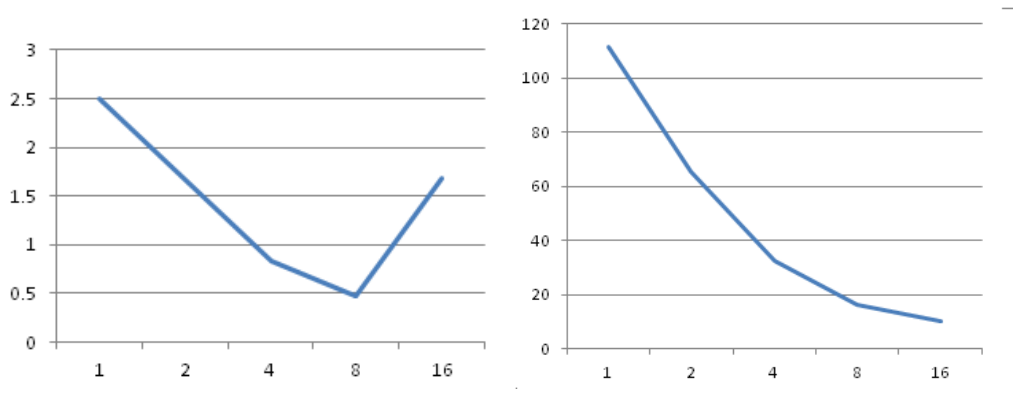
In Table 1 I list the raw timing results (in seconds). I also list the speedups achieved when compared to (1) Matlab as well as the speedup compared to (2) PETSc run with only one processor. Notice that even in the uniprocessor case that PETSc is significantly faster than Matlab. To note, I programmed the Matlab code and the PETSc code to both use sparse matrices. Also, to make the comparison as fair as possible, I have optimized the Matlab code to the best of my abilities using sparse matrices and vectorization and wherever possible. (However, I have not fully optimized PETSc—more speedup to come.)

At first glance you will notice that PETSc is considerably faster than Matlab, even in the serial case. This illustrates, first, how much faster a compiled language (my PETSc application is written in C) is over a scripted language like Matlab. The second detail to notice is that the speedup is nearly linear in the number of processes as long as the problem is large enough. Notice that when $N = 1 \times 10^8$ the speedup from uniprocessor PETSc to PETSc run with 16 processors is 11.12262818. Further test are in order to see how PETSc scales, but we should expect to see nice scaling when we run PETSc with more processors. In this report we have at most used only 16 processors of the approx. 10,000 available at the Fulton Supercomputing lab. (I plan to test on more nodes, but 16 processors is the most I could do without having to let my jobs wait in a queue at Fulton.)

		PETSc, with p the number of processors				
Grid Points	Matlab	p=1	p=2	p=4	p=8	p=16
1×10^6	14.812566	0.809095	0.533468	0.288382	0.470327	0.971024
2×10^6	29.894963	1.681147	1.072255	0.562313	0.370278	1.330877
3×10^6	45.445266	2.502443	1.659407	0.838488	0.467811	1.688088
5×10^6	75.646605	4.451207	2.825604	1.427547	0.782719	0.703109
1×10^7	146.889423	9.041923	5.981552	2.940846	1.57849	1.046028
1×10^8	1496.928579	111.766608	65.19671	32.287323	16.023304	10.048579
1×10^9	***	***	*	*	*	92.058017

TABLE 1. Execution Times. Note: * = timing not yet available, *** = timing not possible—not enough memory! The matrix is a sparse $Ntimes N$.

Also notice that in a few cases presented, such as when $N = 3 \times 10^6$ PETSc is so fast and the problems are so relatively small that the overhead in adding more processes is greater than the speedup achieved. This is due to constaints in network bandwidth and network latency. In parallel programming, these network constraints are usually a central concern. This can be seen in a few of the results below in which the program actually slows down when more processes are added (see the graph in Figure 2a). But, in cases where the problem size is sufficiently large, the scaling is very nice, as that illustrated in Figure 2b.



(A) $N = 3 \times 10^6$. (B) $N = 1 \times 10^8$.

FIGURE 2. Plot of execution times (in seconds, on the y-axis) v. number of processors on the x-axis.

The most important detail, however, to notice is that larger problems are not feasible in single processor applications! Notice the entries in Table 1 marked ***. These are not feasible because of memory limitations—one node has access to at most 128 GB on the particular nodes used in this study. Thus, adding more processors allows us to use memory spread across many nodes.

4. Solving a System of Linear Equation in Parallel

TODO: To come later. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent imperdiet, tellus a rhoncus pellentesque, mi nibh facilisis elit, et interdum magna ligula ut dolor. Pellentesque ac neque ut felis scelerisque malesuada. Quisque quis nisi et nulla rutrum gravida et ut mi. Donec lorem libero, tincidunt sit amet dapibus non, venenatis at lacus. Etiam aliquet bibendum neque at adipiscing. Vivamus turpis purus, consequat eu pharetra id, sollicitudin non purus. Praesent ullamcorper tempor ligula quis egestas. Nunc in sem neque. In vel metus lorem.

5. Conclusion

TODO: Conclusion to come later.

Nunc quam libero, sollicitudin eget pulvinar nec, fringilla vel dolor. Vestibulum egestas consequat mi, nec semper metus malesuada eget. Donec vulputate purus vel eros eleifend vitae sodales risus pellentesque. In eget molestie lacus. Sed suscipit ullamcorper arcu, at condimentum elit blandit et. Proin tristique lectus sed augue dictum ac convallis tellus pellentesque. Aenean eget diam vel eros euismod placerat eget ac quam. Duis facilisis, diam at sagittis congue, nulla purus mattis augue, at ullamcorper dolor purus accumsan libero. Praesent viverra pulvinar felis a convallis. Cras vel nulla id justo tempor varius ullamcorper sed diam. Nullam nec massa et erat viverra fermentum quis a sapien. Mauris ut libero erat, a mattis sapien. Etiam id sollicitudin arcu. Nam lacinia quam quis leo malesuada sodales.

Mauris vitae arcu eget sapien consequat blandit. Sed justo sem, laoreet vitae semper nec, posuere eget turpis. Pellentesque ac eros vitae turpis commodo congue ut vel nisl. Donec malesuada arcu et risus mattis ac ullamcorper magna hendrerit. Donec imperdiet mollis pulvinar. Nullam vulputate iaculis cursus. Curabitur urna lacus, semper vitae auctor in,

luctus sit amet velit. Mauris mollis ante quis urna feugiat sed tincidunt arcu rutrum. Mauris vel ante ligula. Nunc non diam vitae metus rhoncus tincidunt nec at magna. Lorem ipsum dolor sit amet, consectetur adipiscing elit.

References

[1] Inside story: Parallel bars. *The Economist*, June 2011.

[2] Stphane Adjemian, Houtan Bastani, Michel Juillard, Ferhat Mihoubi, George Perendia, Marco Ratto, and Sbastien Villemot. Dynare: Reference manual, version 4. *Dynare Working Papers*, 1, 2011.

[3] Eric M. Aldrich, Jess Fernndez-Villaverde, A. Ronald Gallant, and Juan F. Rubio-Ramrez. Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors. *Journal of Economic Dynamics and Control*, 35(3):386393, 2011.

[4] Michael Creel. User-friendly parallel computations with econometric examples. *Computational Economics*, 26(2):107128, 2005.

[5] Jurgen A. Doornik, Neil Shephard, and David F. Hendry. *Parallel Computation in Econometrics: A Simplified Approach.* Number 184. Chapman & Hall/CRC, 2006.

[6] Christopher Ferrall. Solving finite mixture models in parallel. *Computational Economics*, 303003, 2003.

[7] Christopher Ferrall. Solving finite mixture models: Efficient computation in economics under serial and parallel execution. *Computational Economics*, 25(4):343379, 2005.

[8] Edward Herbst and Frank Schorfheide. Sequential monte carlo sampling for DSGE models. SSRN Scholarly Paper ID 2182710, Social Science Research Network, Rochester, NY, November 2012.

[9] Kenneth L. Judd. *Numerical methods in economics*. The MIT press, 1998.

[10] Morton I. Kamien and Nancy L. Schwartz. *Dynamic optimization: the calculus of variations and optimal control in economics and management*. Courier Dover Publications, 2012.

[11] E. J. Kontoghiorghes. Parallel strategies for computing the orthogonal factorizations used in the estimation of econometric models. *Algorithmica*, 25(1):58–74, May 1999.

[12] Christopher A. Swann. Maximum likelihood estimation using parallel computing: An introduction to MPI. *Computational Economics*, 19(2):145–178, April 2002.

E-mail address: jmbejara@gmail.com