A Note on Parallelizing the Parameterized Expectations Algorithm

Michael Creel[*]

June 15, 2005

Abstract

The parameterized expectations algorithm (PEA) involves a long simulation and a nonlinear least squares (NLS) fit, both embedded in a loop. Both steps are natural candidates for parallelization. This note shows that parallelization can lead to important speedups for the PEA. I provide example code for a simple model that can serve as a template for parallelization of more interesting models, as well as a download link for an image of a bootable CD that allows creation of a cluster and execution of the example code in minutes, with no need to install any software.

Keywords: parameterized expectations; parallel computing

JEL classifications: E27, C63, C88

[*]Department of Economics and Economic History, Edifici B, Universitat Autònoma de Barcelona, 08193 Bellaterra (Barcelona) Spain. *email*: michael.creel@uab.es; *Tel.:* 93-581-1696; *FAX*: 93-581-2012. I am very grateful to Javier Fernández Baldomero for his help in getting started with MPITB. This research was supported by grants SGR2001-0164 and SEC2003-05112.

The parameterized expectations algorithm (PEA) is a well-known method of solving nonlinear stochastic dynamic models with rational expectations (Marcet and Marshall, 1994; Marcet and Lorenzoni, 1999; Christiano and Fisher, 2000). There are a number of different implementations which use different methods of weighting residuals and of computing integrals (see Christiano and Fisher, 2000, Table 1). Marcet (1988) implemented what Christiano and Fisher refer to as a conventional PEA. This version replaces the expectation function of unknown form with a parametric approximating function. A long series of simulated data is generated, taking the parameters of the approximating function as given. Next, the approximating function is fitted to the generated data to update the parameters. The algorithm repeats these steps until the parameters no longer change, at which point convergence occurs. If the simulation is long enough so that the fitted parameters at convergence are the same regardless of the particular sequence of random numbers used for the simulation, the fitted approximating function may be treated as a nonstochastic object. By choosing the approximating function appropriately and using a long enough simulation, the PEA solution can be made arbitrarily close to the exact rational expectations solution (Marcet and Marshall, 1994).

The conventional PEA has the advantage that it is a simple algorithm that is easy to understand, and it has a clear theoretical justification. A disadvantage is that it may be necessary to use an extremely long simulation in order to obtain the same fitted coefficients of the approximating function across replications of the simulation. This can cause the algorithm to be quite computationally demanding (see Christiano and Fisher, Table 4, for example). Christiano and Fisher find that alternative versions of the PEA seem to offer accuracy as good as that of the conventional PEA, with much less computational cost.

However, these findings are based on examples rather than theoretical results, so it is not clear that they are general. As such, the conventional PEA remains an important tool. This note shows that the conventional PEA is easily modified to run in parallel on multiple CPUs. Both the long simulation and the estimation step to update the parameters can be done in parallel. If enough CPUs are used, this can greatly reduce the time needed to find a solution to a given level of accuracy.

1 Parallelizing the PEA

Maliar and Maliar (2003) discuss a moving bounds method of imposing stability on the conventional PEA, that avoids explosive behavior due to poor choices of initial parameter values. They also provide simple example code written in the MATLAB[1] language. The problem they solve is simple and illustrative, which suits the present purpose. If this problem can be solved more quickly in parallel, then more interesting and computationally costly problems will also benefit.

In the first step, the conventional PEA calculates expectations by Monte Carlo, using a long simulation that generates data, conditional on the parameters of the approximating function. Since the model is assumed to have a stationary and ergodic distribution, expectations computed using a single long simulation are equivalent to the ensemble average of expectations computed using a number of shorter length simulations. This note proposes to calculate each of the shorter simulations in the ensemble on a different CPU. The second step of the algorithm is a nonlinear least square (NLS) fit of the expectations

[1] TM the Mathworks, Inc.

function to the generated data. The NLS estimation step exhibits data parallelism, in that the data used to evaluate the objective function can be broken into a number of blocks, and the contributions of each block to the overall criterion function can be calculated independently of one another on different CPUs. This idea is presented in more detail for the MLE and GMM estimators in Creel (forthcoming).

GNU Octave[2] is a freely available high-level matrix programming language that has a syntax that is compatible with MATLAB's. Most MATLAB scripts will run unmodified under Octave. The code by Maliar and Maliar will run using Octave, once the proprietary nonlinear least squares routine used to update the parameters of the approximation is replaced by a freely available alternative. But modification is needed to make the code execute in parallel.

The Message Passing Interface (Message Passing Interface Forum, 1997) is a specification of a mechanism for passing instructions and data between different computational processes, which may run on different nodes of a cluster of computers. This specification has been implemented in a number of packages, including LAM/MPI (LAM team, 2004). LAM/MPI provides a library of C and FORTRAN functions that allow one to write MPI parallel programs, along with support programs to use the functions. To make direct use of the libraries, one must program in C, C++, or FORTRAN. GNU Octave, in common with most similar languages, offers a means of linking in compiled FORTRAN, C and C++ code. Fernández Baldomero *et al.* (2004) have written the MPI toolbox (MPITB) for GNU Octave. This is a collection of binding functions and support scripts that allow MPI functions to be used in Octave code that can run in parallel on a cluster of computers.

[2] `www.octave.org`

Using this toolbox, a parallelized version of the PEA was written, using Maliar and Maliar's code as a guide. For an overall simulation of length T, each of N computers performs T/N (rounded up to the nearest integer) simulations, and each computer contributes to the calculation of the NLS objective function in the minimization step. The example code uses the same approximating function and model parameters as do Maliar and Maliar (2003).

GNU Octave is an interpreted language, and it is not particularly fast to evaluate loops. For a computationally demanding problem like the PEA, it makes little sense to parallelize Octave code that contains loops, since similar or better performance could be obtained by using a compiled language and doing calculations on a single CPU. For this reason, the PEA simulation loop was written in C++ and linked in dynamically, in the same way this was done by Fernández *et al.* to create the MPITB binding functions. The resulting mixed Octave/C++ code has a convenient, user-friendly interface, but it is also efficient, since Octave is used where vectorization is possible, but C++ is used for evaluation of loops.

The example code used to generate the results that follow is available from the author upon request. To use the code, one must have GNU Octave and MPITB installed, as well as additional supporting code for BFGS minimization, and one must have a properly configured cluster of computers. All of the software is available pre-installed and ready to run on the ParallelKnoppix bootable CDROM described in Creel (forthcoming). An image of the CDROM can be downloaded from `pareto.uab.es/mcreel/ParallelKnoppix`[3]. With this CDROM, one can create a temporary, non-destructive cluster in minutes, and

[3]All of the computer code mentioned in this note is in the `~/Desktop/ParallelKnoppix/Examples/Octave/pea` directory of the CDROM.

can then experiment with the code without having to install anything. When the cluster is shut down, the computers return to their original state, with their original operating system unaffected.

2 Performance

To investigate the impact of parallelization on time needed to run the code, we need to choose T, the number of simulations. A small value of T will result in little speedup from parallelization, since communication overhead and the time needed to compute the non-parallelized portions of the code will form a relatively large portion of the total computational time. When T is larger, the parallelized portion consumes relatively more time, and the speedup from parallelization is greater[4].

The basic requirement for choosing T is that it must be large enough so that the results of interest don't change in any important way if the algorithm is run another time. While one could check various moments of the stationary distribution of the model, which is the object of fundamental interest, I instead use the crude but simple criterion that the coefficients of the function that approximates expectations should be the same at the second decimal place across all replications of the PEA solution that use different sequences of random numbers in the simulations. Two successive runs of the program reveal that this does not occur when $T = 100,000$. If a good speedup can be obtained with this length of simulation, a better speedup will be obtained with a longer simulation that would be needed to obtain the required accuracy. So showing a good speedup from parallelization with this length of simulation is sufficient.

[4]The relationship between the attainable speedup that can result from parallelization and the proportion of the code that is parallelizable is known as Amdahl's law (Amdahl, 1967).

The cluster used for the timings was a homogeneous cluster of Pentium IV machines that run at 2.8 GHz, each with 1MB of level 2 cache. Each computer has a 3COM 3c905 Tornado network card, and they were connected with a dedicated 100MB/s ethernet network using a 3COM OfficeConnect dual speed switch. The cluster was created using the ParallelKnoppix CDROM that is mentioned above.

Table 1 contains timing results when different numbers of computers are used to solve the model. The timings are on a per iteration basis. Different sequences of random numbers were used for each run, which explains the minor irregularities in timings, since the number of iterations needed for the NLS estimation step to converge varies with the technology shocks that are drawn. We can see that parallelization has a very notable effect on performance. The PEA simulation step runs 5.71 times faster using 10 nodes than it does using 1 node. The NLS estimation step achieves a speedup of 5.54 times, and the overall speedup is 5.47 times. We also see that the runtime is still declining as we move from 9 to 10 nodes, so use of additional nodes would further reduce computational time. It bears emphasis that the example problem is very simple and executes fairly rapidly on a single CPU (around 35 iterations are needed to achieve convergence, depending upon the technology shocks that are drawn, so the serial execution time to convergence is about 2 minutes on a Pentium 4 CPU running at 3 GHz). For more complicated problems with more complicated approximating functions that use more parameters, the speedup would be greater, since a greater proportion of total runtime would be concentrated in the parts of the code that are parallelized.

3 Conclusion

This note has shown that parallelization can give an important reduction in the time needed to solve a model using the PEA. The model used here is extremely simple, but nevertheless a good performance improvement is obtained from parallelization. For more complicated models that are more costly to simulate, or for approximating functions with more parameters, better improvements could be obtained using larger clusters. It is also worth mentioning that extremely long simulations require considerable amounts of memory, particularly to do the NLS step. When parallelized code is used, the data used to calculate the NLS objective function resides on a number of computers. Thus, the memory requirements of each machine in a cluster are more modest than what would be needed to perform the calculations on a single computer.

The code that accompanies this note can serve as a template for writing parallel Octave code for more interesting and computationally demanding models. Since the MPITB binding functions respect the general MPI specification, the provided Octave code serves as a useful model for parallelizing existing PEA codes written in C and FORTRAN.

Table 1: Timings (sec.) per iteration, $T = 100{,}000$

nodes	PEA simulation	NLS estimation	Total
1	0.16	3.44	3.61
2	0.089	1.48	1.58
3	0.064	1.15	1.22
4	0.052	0.93	0.99
5	0.044	0.93	0.98
6	0.039	0.80	0.85
7	0.033	0.70	0.75
8	0.032	0.75	0.79
9	0.029	0.73	0.77
10	0.028	0.62	0.66

References

[1] Amdahl, G. (1967), Validity of the single processor approach to achieving large-scale computing capabilities, AFIPS Conference Proceedings, **30**, 483-485.

[2] Christiano, L. and Fisher, J. (2000), Algorithms for solving dynamic models with occasionally binding constraints, *Journal of Economic Dynamics and Control*, **24**, 1179-1232.

[3] Creel, M. (forthcoming), User-friendly parallel computations with econometric examples, *Computational Economics*.

[4] Fernández Baldomero, J. *et al.*, (2004) MPI toolbox for Octave, VecPar '04, Valencia, Spain, June 28-30 2004, http://atc.ugr.es/~javier/investigacion/papers/VecPar04.pdf

[5] LAM team (2004) LAM/MPI parallel computing, http://www.lam-mpi.org/.

[6] Maliar, L. and Maliar, S., Parameterized expectations algorithm and the moving bounds, *Journal of Business and Economic Statistics*, **21**, 88-92.

[7] Marcet, A. (1988), Solving non-linear stochastic models by parameterizing expectations, working paper, Carnegie Mellon University.

[8] Marcet, A. and Lorenzoni, G. (1999), The parameterized expectations algorithm: some practical issues, in *Computational Methods for Study of Dynamic Economies*, R. Marimon and A. Scott, eds., New York, Oxford University Press, pp. 143-171.

[9] Marcet, A. and Marshall, D. (1994), Solving nonlinear rational expectations models by parameterized expectations: convergence to stationary solutions, Federal Reserve Bank of Minneapolis Discussion Paper 91.