Economic Modelling 52 (2016) 3–12

Contents lists available at ScienceDirect

Economic Modelling

journal homepage: www.elsevier.com/locate/ecmod

Solving intertemporal CGE models in parallel using a singly bordered block diagonal ordering technique ☆

Pham Van Ha [a], Tom Kompas [a,b,*]

[a] Australian Centre for Biosecurity and Environmental Economics, Crawford School of Public Policy, Australian National University, Australia
[b] Centre of Excellence for Biosecurity Risk Analysis, University of Melbourne, Australia

ARTICLE INFO

Article history:
Accepted 14 July 2015
Available online 28 August 2015

Keywords:
Singly bordered block diagonal ordering
parallel computing
Intertemporal CGE models

ABSTRACT

The paper introduces a direct ordering method that employs a special feature of an intertemporal Computable General Equilibrium (CGE) model to reorder its first-order partial derivative matrix into a Singly Bordered Block Diagonal (SBBD) form. The matrix can then be decomposed into LU form and solved in parallel. With this method, the numerical results from the paper show a substantial advantage in computational time and memory use for parallel solutions of intertemporal CGE models in comparison to current serial solution methods. A solution for an intertemporal and regional model of the Vietnamese economy is provided as an example and comparator for the different methods.

1. Introduction

CGE models are large systems of nonlinear equations. Intertemporal CGE models are even larger since they must be solved over several time periods, potentially spanning a very long time horizon. Direct search methods are a poor solution choice for these large-scale CGE models due to their lack of robustness and the resulting curse of dimensionality. The secant or Newton iterative methods, which depend on a first-order derivative matrix (either by symbolic or finite difference approximation), are therefore more practical choices for CGE solver software packages. Finding a solution for CGE models, in other words, essentially involves finding a way to handle and solve large first-order partial derivative matrices.

Over the years, a number of good linear algebra packages have been built and professional software packages have been constructed offering dedicated solutions to CGE models, such as GEMPACK and GAMS. Nevertheless, the growing size or dimension of CGE models, especially intertemporal CGE models, remains a serious challenge for current CGE solution methods.

This paper addresses this challenge by proposing a direct reordering method for the first-order partial derivative matrices that arise from CGE model solutions. The ordering method facilitates a parallel solution and provides a substantial advantage in computational time and memory use for the solution of intertemporal CGE models.

Our contribution is also important in the way that it facilitates the building of larger, more detailed intertemporal CGE models. The lack of efficient solution methods has hindered the popularity of intertemporal CGE models and where they are deployed the size of the intertemporal components is usually sacrificed in favour of a finer time grid and more precise solutions.

The organisation of the paper is as follows. In Section 2 we briefly summarise the current literature on general equilibrium modelling and its computational challenges. Section 3 details current methods and the software available for the solution of CGE models. Section 4 discusses how the direct method can be made more efficient with a Singly Bordered Block Diagonal matrix. Section 5 follows this logic and discusses how a Jacobian matrix of intertemporal CGE models can be ordered into SBBD form. Section 6 presents numerical results for the parallel solution of a CGE model based on our direct ordering method, providing an example and comparator across solution methods of an intertemporal and regional model of the Vietnamese economy. Section 7 concludes.

2. Computable general equilibrium models and their computational challenges

CGE models are a model of the economy as a whole, with multiple agents (i.e., producers, consumers, government) interacting in different markets. CGE models were originally built on the foundations of a Walrasian General Equilibrium economy, and their computable property relies on the uniqueness and stability of the Walrasian approach. Practical CGE models can be represented and solved as non-linear programming problems (see Dixon and Parmenter (1996)), however, we find that most CGE models, in practice, are normally represented as a system of nonlinear equations with the agents' (i.e., consumers,

☆ Acknowledgement: Based on a paper first presented at the 3rd International Symposium in Computational Economics and Finance (ISCEF), April, 10–12, 2014, Paris, with thanks to participants for helpful comments.

* Corresponding author at: Australian Centre for Biosecurity and Environmental Economics, Crawford School of Public Policy, Australian National University, Australia.
E-mail addresses: ha.pham@anu.edu.au (P. Van Ha), Tom.Kompas@anu.edu.au (T. Kompas).

http://dx.doi.org/10.1016/j.econmod.2015.07.011
0264-9993/

producers, etc.) optimization problems being solved analytically in the form of demand and supply equations. The resulting nonlinear system is calibrated with actual economic data and if the solution of the system is unique and stable it can be found, in principle, by the convergence of an iterative solution method such as the now 'old fashioned' Newton–Raphson method.

The rapid increase in practical CGE modelling in the second half of 20th century was supported by the important work of Leontief (1936), who introduced the first input–output table of the US economy. The building of input–output tables was later facilitated by the introduction of the System of National Accounts in 1953 (Nations, 1953), and its latest version in 2008 (EC et al., 2009) by the United Nations. The System of National Accounts has been gradually adopted by many countries since and provides the essential data for CGE modelling.

The model accepted by many CGE modellers as the first CGE model was the multi-sector model of Norwegian economy by Johansen (1960). The model had 22 production sectors and one representative consumer. Johansen's (Johansen, 1960) model was the first economy-wide model of separate agents (i.e., consumers and producers) instead of through a single agent as in previous works (Dixon and Jorgenson, 2013). Johansen's (Johansen, 1960) contribution was not limited to building the first multi-agent CGE model, he also proposed a single step 'linearization approximation' solution method for CGE models. The idea was to linearize the system of non-linear equations in an approach similar to finding a solution of a system of first-order Taylor-series expansions of non-linear equations (see Section 3.1 for further details). The method was later described by Dixon and Parmenter (1996) as the 'derivative approach', to be distinguished with the 'iterative method'. The method has been further developed and used by GEMPACK, a popular CGE solver package (see (Harrison and Pearson (1996) and Dixon and Parmenter (1996))).

The second stream of CGE modelling, following Johansen (1960), begins with the applied general equilibrium works by Shoven and Whalley (1972, 1973), who used a search algorithm to compute the solution for a Walrasian economy designed by Scarf (1967) and Scarf and Hansen (1973). The idea of the numerical algorithm is to search for a fixed point equilibrium of the Walrasian economy by creating a fine grid of augmented price vectors on a unit simplex, moving from one convex hull created by the points on the grid to another, until finding the one convex hull that can represent the fixed point (see, (Scarf (1967), Scarf and Hansen (1973), and Shoven and Whalley (1973))). Although the algorithm is reliable and can find the fixed point in a finite number of steps, it is not very efficient and has lost its popularity when dealing with models capable of forecasting and answering practical policy questions (Dixon and Parmenter, 1996).

In the latter quarter of 20th century, with the introduction of many efficient CGE solver packages, the relatively inefficient single step solution approximation as well as Scarf's method was gradually replaced by other more efficient solution methods, the GAMS's iterative method and GEMPACK's linearisation method (see Section 3).

With the improvement in computing capacity the size of CGE models is growing rapidly. Modern CGE software packages like GEMPACK and GAMS are written in the form of matrix languages, which define the model's dimensions as sets of goods, industries, household types, occupations, countries, and so on, over time. The size of these inputs is limited only by the available input data. This enormous increase in size and complexity poses many challenges for solution packages.

Being built up from millions of static equations, the introduction and use of dynamic CGE models poses even greater challenges to CGE model solvers. There are some available shortcuts. The easiest treatment of large multi-equation problems is to break down the dynamic CGE models into several time segments, and solve them sequentially. Recursive dynamic CGE models are used exactly for this purpose. Recursive models assume each agent has static or myopic expectations, or that they optimise their behaviour within one period only. The model is divided into two components. First, a 'within period' static CGE model is solved with myopic expectations. Second, the 'between-period' model is then solved by updating certain exogenous (or 'within-period' component) variables such as the capital stock, labour, and productivity parameters. These exogenous variables connect within-period components through time (Devarajan and Robinson, 2013; Scollay and Gilbert, 2000). The size of recursive dynamic CGE model is therefore much smaller than solving a multi-period model at once. The problem, of course, is that agents are not being able to look forward in this setting.

In theory, a recursive model can be revised to incorporate forward looking behaviour by guessing and employing a 'shooting method' (Dixon et al., 1992, pp. 334-6). The model still can be solved sequentially, but the number of adjustments (after each shot) can be large. With practical CGE models involving hundred of industries, the shooting method is proving to be computationally expensive (Dixon and Rimmer, 2002). The MONASH (Dixon and Rimmer, 2002) model, a successor of ORANI model (Dixon et al., 1982), is a typical example of a recursive model. This model later has been modified into a USAGE model and applied for the US economy by Dixon and Rimmer (2004).

Unlike recursive treatments, intertemporal CGE models attempt to solve multi-period settings directly. Intertemporal CGE is designed as an extension of a static CGE model and often includes elements of neoclassical growth theory (Devarajan and Robinson, 2013). Solutions for intertemporal CGE models are based on stable saddlepath equilibrium properties. For the model to be solved numerically, intertemporal motion equations are represented in finite difference form (see Dixon et al. (1992), pp. 340-8). Finite differenced intertemporal equations, together with the intratemporal (intra-period) static CGE model and steady state conditions, are then solved simultaneously to complete the model. Solutions of intertemporal CGE models are computationally demanding. Examples of intertemporal CGE models are the global intertemporal model built by McKibbin (1987) and McKibbin and Sachs (1991) and the intertemporal CGE model for Australian economy, or ORANI-INT (Malakellis, 1998).

The purpose of our paper is to further tackle the numerical and computational challenge of intertemporal CGE models. Even though intertemporal CGE models have the advantage in allowing fully forward looking behaviour, they are dimensionally very large, and with the inclusion of multiple one-period static CGE equations, they are difficult to solve without limiting assumptions. We propose a method that allows part of these one-period static CGE models to be solved in parallel, hence reducing computational time overall. Our main contribution is to solve the first-order partial derivative matrix of intertemporal CGE models in parallel, thus providing computational benefits to both the iterative and linearisation methods, as the former needs to solve a first-order derivative matrix to update the solution after each step and the latter needs to solve a matrix to approximate a new solution.

3. Current software packages for CGE models and solution methods

There are three main software packages that are dedicated to solving CGE models: GAMS, MPSGE, and GEMPACK (Horridge et al., 2013). Since MPSGE is only a sub-system in GAMS, providing syntax to define CGE models and preform some pre-processing work for GAMS, it's best to think of only two main packages, GAMS and GEMPACK. Both software packages have advantages and disadvantages in terms of their solution methods for CGE models.

In addition to these two popular software packages that specialise in solving more general CGE models, there are also other tailor-made packages that are written to solve specific CGE problems. A typical example of such a package is the MSG algorithm (McKibbin, 1987; McKibbin and Sachs, 1991), which is used to solve MSG and G-CUBED models (see McKibbin and Sachs (1991)). Although we will not be able to compare our method with the MSG algorithm directly, because

it is written to solve specific medium size models only [i.e., the G-CUBED model employs 5000 equations over 100 years or half a million equations in total, see McKibbin and Wilcoxen (1999)], we review the algorithm for the sake, at least, of finding any similarity with our direct reordering solution approach.

3.1. The GEMPACK linearization method

GEMPACK employs four different solution methods: Johansen, Euler, Gragg, and Midpoint (Harrison and Pearson, 1996). All the above methods share the same property. They are linear approximations with different extrapolation techniques. The GEMPACK solution method can be represented simply as a first-order Taylor series expansion of a vector of multivariate functions $f(x, y) = 0$:

$$f_x dx = f_y dy \tag{3.1}$$

where x, y are a vector of endogenous and exogenous variables. The system, given a point (x_0, y_0), is then shocked with a small deviation $dy = y_1 - y_0$. The solution for a given point x, after a small change or shock, can be easily found by the inversion of matrix f_x:

$$x_1 = x_0 + f_x^{-1} f_y dy. \tag{3.2}$$

The approximation is good for a small change in y, or simply, when dy is small. When dy is large, which is usually the case, dy can be divided into smaller steps: y changes from y_0 to y_1 then to y_2 … to y_N. The solution for x in these steps can be found by a recursive solution of the system (i from 1 to N):

$$x_i = x_{i-1} + f_x^{-1}|_{\substack{x = x_{i-1} \\ y = y_{i-1}}} f_y|_{\substack{x = x_{i-1} \\ y = y_{i-1}}} dy_i. \tag{3.3}$$

Thus, by dividing the shock y_0 to y_N into smaller steps, we can obtain a better approximation for x_i and finally x_N (see, Harrison and Pearson (1996)).

The solution of a CGE model after a shock can be done with a 1 step (or Johansen) method, and a 2 and 4 steps methods where solutions can be denoted as x(1),x(2) and x(4). To enhance the precision we can form the final solution as a Richardson extrapolation of, say, x(1), x(2) and x(4) (named after Richardson (1911)).

Unlike iterative solutions, which refine the solution after each step, the GEMPACK's linearization method allows the user to choose how many steps can be performed, hence the user can have better control over solution time and accuracy. Thanks to this feature, GEMPACK is usually the fastest CGE solver available. But speed comes with a price, in the sense that the user cannot guarantee that the solution obtained from the linearization method converges. That said, GEMPACK does provide results for simulation accuracy in terms of checks for convergence.

To solve a (potentially large) first-order derivative matrix f_x^{-1}, GEMPACK uses the direct serial solver HSL 2013's (HSL, 2013) MA48 (or MA28, depending on the user's choice). These solvers were written in Fortran and have been using as a benchmark for other direct non-symmetric linear solvers. Therefore, it is perhaps not surprising that GEMPACK has proven to be the best performer [see Table 20.7 in Horridge et al. (2013)] among the software packages available for CGE modelling.

3.2. The GAMS's iterative methods

GAMS is a truly flexible system. It is designed for big optimization problems. However, CGE models can also be solved easily in GAMS using a dummy optimum function. In essence, GAMS uses a direct system of equations solver to solve CGE models. The GAMS solvers that are frequently used for CGE models are PATH, MILES or optimiser MINOS (see, Horridge et al. (2013)).

PATH (Dirkse and Ferris, 1995) and MILES (Anstreicher et al., 1992) are part of the GAMS Mixed Complementarity Problem solver that is available with any GAMS system. The two solvers are similar although PATH is faster and more robust than MILES (Billups et al., 1997). We thus set MILES aside for our purposes. PATH is a Newton-based solver, which updates the solution by solving the system involving a Jacobian matrix of non-linear systems (Dirkse and Ferris, 1995). In our case, the non-linear system is the CGE Model. In a way similar to GEMPACK, and since the Jacobian matrices in CGE models are usually very large, we are faced with the same challenge as GEMPACK. At its core, PATH uses LUSOL (Gill et al., 1987), as a linear solver to solve the system of Jacobian matrices. Since LUSOL is a serial direct linear system solver, it suffers from the same disadvantages as MA48.

Unlike PATH and MILES, MINOS (see, Murtagh and Saunders (1982), for a detailed description) is an optimisation solver, not a direct non-linear system solver. An example of using a MINOS solver to solve a CGE model is the CGE model for property tax analysis in Idaho (Julia-Wise et al., 2002). Because MINOS is a nonlinear programming solver, the CGE model can be represented as a system of non-linear constraints and a dummy objective function can be formed to satisfy the MINOS input requirements. A set of variables that satisfies all of the constraints will be the optimum.

With nonlinear constraints, MINOS employs a projected Lagrangian algorithm, which involves a sequence of major iterations. Each iteration will require the solution of a linearly constrained sub-problem, which contains linearized versions of the nonlinear constraints, or the Jacobian matrix (Murtagh and Saunders, 1982). Solving the linearly constrained sub-problems, in turn, requires solving a linear system involving a Jacobian matrix. Again, MINOS uses LUSOL to perform this task.

3.3. The MSG algorithm

Unlike the above two software packages that try to solve CGE models by inverting the entire first-order derivative matrices, the MSG algorithm tries to exploit the special feature of intertemporal CGE models to solve the problem faster. The algorithm has been described in detail in McKibbin (1987) and McKibbin and Sachs (1991). Here, only a brief discussion will be provided.

McKibbin (1987) and McKibbin and Sachs (1991) represent an intertemporal CGE model as a system of nonlinear difference and nonlinear equations:

$$X_{t+1} = \Phi_1(X_t, e_t, Z_t, E_t) \tag{3.4}$$

$$Z_t = \Phi_2(X_t, e_t, Z_t, E_t) \tag{3.5}$$

$$e_{t+1} = \Phi_3(X_t, e_t, Z_t, E_t) \tag{3.6}$$

where X_t, e_t, Z_t, E_t are the state, jump, endogenous and exogenous variables respectively. To solve system, the first step involves a linearization around some point, usually either the steady state or on the transition path to steady state using a first-order Taylor expansion. The linearized system will then become a system of mixed linear deference equations and linear equations:

$$\overline{X}_{t+1} = a_1\overline{X}_t + a_2\overline{e}_t + a_3\overline{Z}_t + a_4\overline{E}_t \tag{3.7}$$

$$\overline{Z}_t = b_1\overline{X}_t + b_2\overline{e}_t + b_3\overline{Z}_t + b_4\overline{E}_t \tag{3.8}$$

$$\overline{e}_{t+1} = c_1\overline{X}_t + c_2\overline{e}_t + c_3\overline{Z}_t + c_4\overline{E}_t \tag{3.9}$$

where $\overline{X}_t, \overline{e}_t, \overline{Z}_t, \overline{E}_t$ denote the deviation from linearisation point, and a_i, b_i, c_i are the first-order derivative matrices of Φ_j.

In the next step, the endogenous variables $\overline{Z}_t$ will be substituted out to condense the system into a system of linear difference equations only, or:

$$\overline{X}_{t+1} = a'_1\overline{X}_t + a'_2\overline{e}_t + a'_3\overline{E}_t \tag{3.10}$$

$$\overline{e}_{t+1} = c'_1\overline{X}_t + c'_2\overline{e}_t + c'_3\overline{E}_t. \tag{3.11}$$

The above linear system is solved by first assuming an arbitrary terminal period T as a steady state, so that, according to Eq. (3.11):

$$\overline{e}_T = H_{1T}\overline{X}_T + H_{2T}\overline{E}_T \tag{3.12}$$

where H_{1T}, H_{2T} are time sub-scripted matrices. Going back one period, we can solve for e_{T-1}:

$$H_{1T}\overline{X}_T + H_{2T}\overline{E}_T = c'_1\overline{X}_{T-1} + c'_2\overline{e}_{T-1} + c'_3\overline{E}_{T-1} \tag{3.13}$$

and substituting Eq. (3.10) into (3.13) and rearranging gives:

$$\overline{e}_{T-1} = H_{1T-1}\overline{X}_{T-1} + H_{2T-1}\overline{E}_{T-1} + C_{5T}\overline{E}_{T-1}. \tag{3.14}$$

Repeating this exercise further we obtain:

$$\overline{e}_t = H_{1t}\overline{X}_t + H_{2t}\overline{E}_t + C_{5T}\{\overline{E}_{t+1}\ldots\overline{E}_T\}. \tag{3.15}$$

As designed, the jump variable will be a function of the current state variable and (current and future) exogenous variables only. McKibbin (1987) and McKibbin and Sachs (1991) repeat the above procedure until H_{1t}, H_{2t} become independent from the choice of T, and the system is then (as they claim) on a stable manifold. After the rule linking $\bar{e}_t$ and $\overline{X}_t$ is performed, the linear difference system can be solved forward for any period t using the rule given by Eq. (3.15). The endogenous variables $\overline{Z}_t$ can also be recovered once $\bar{e}_t$ and $\overline{X}_t$ have been found.

By separating the solution of intra-period variables (substituting $\overline{Z}_t$) and inter-period variables ($\bar{e}_t$ and $\overline{X}_t$) with a nicely designed difference equations solver, the MSG algorithm is expected to be faster in solving an intertemporal CGE model than inverting large matrices as in the case of GAMS and GEMPACK. However, the MSG algorithm also has drawbacks: it is not a parallel solver and it has to rely on a first-order Taylor expansion to approximate the solution. The accuracy of the algorithm is also in doubt when the system is highly nonlinear, although McKibbin and Sachs (1991) assure its accuracy for the MSG model.

In summary, the GAMS solution method is based on an iterative mechanism to find a solution (using either a direct nonlinear system solver or a nonlinear programming solver). The number of iterations needed to find a solution will depend on the specific problem to be solved. The computing time, therefore, could be significantly higher compared to GEMPACK, which limits the number of times the linear system solver is employed by defining how refined are the solution steps. In any case, it seems clear that convergence in GAMS is sound, while in the GEMPACK convergence and the precision of the solution depends on how small the shock will be as divided between solution steps. Both packages, however, rely on a serial matrix solver in each solution step to solve CGE models, hence will be inefficient with large intertemporal CGE models. The MSG algorithm, on the other hand, exploits the special feature of intertemporal CGE models to solve them faster, but its accuracy is a concern as it relies on a first-order Taylor expansion only.

3.4. Parallel solution

As we can see from the discussion above, solving CGE models using either GEMPACK, GAMS or special purpose packages will involve solutions of large linear systems. Solving them on a serial computer is expensive both in memory usage and computing time. Parallel solutions are simply the next best logic step.

The idea of a parallel solution for CGE models was first introduced in GEMPACK version 10 (Horridge and Pearson, 2006). However, GEMPACK only uses parallel processes to solve the same CGE model with different shocks. That means every single solution of the model must be carried out serially, and the method can thus use only limited parallel resources to speed-up computation time or enhance memory.

There are many software libraries available for parallel solutions of large matrices, but none of the current CGE solver software packages are yet to employ parallel matrix solvers. This is due partly to the fact that many current parallel solvers are not adequate for the very large non-symmetric matrices produced by an intertemporal CGE model. The available iterative solvers are also not reliable while direct solver methods (e.g., the LU decomposition) cannot efficiently solve the curse of dimensionality. Current direct solvers depend on matrix ordering packages to reduce the number of the 'fill in' elements in the factorisation phase of LU decomposition process in order to overcome the curse of dimensionality. However, in many cases the ordering phase consumes more memory and computing time than the LU decomposition itself (Hu and Scott, 2005).

In this paper we show that the first order partial derivative matrix derived from intertemporal CGE models can be transformed into SBBD form directly. In addition, when we construct the matrix in this way we can skip the matrix ordering phase. Most importantly, a special feature of the intertemporal CGE model allows a very efficient matrix ordering into SBBD form, which we show to be vital for efficient parallel computing.

4. SBBD matrix and the direct method for solving linear systems

Solving a large matrix can be done by either a direct or iterative method. While the direct method involves a LU decomposition of matrices, the iterative method will refine the solution after a series of steps that involve multiplication of sparse matrices and vectors, with no matrix inversion. The iterative method has proven its advantages (with restrictions) in the symmetric case but is not robust for a general case. For normal equations (i.e., non-symmetric equations), the method does not guarantee success, especially in the case of a 'poorly conditioned' matrix (see, for example, Saad (2003)).

The direct method, on the other hand, is more robust and more reliable in the general case. Nevertheless, the direct method requires much more computing resources due to the 'fill-in' problem (i.e., the matrix to be solved can be very sparse, but the lower and upper matrices can be dense, where the extra elements in the lower and upper matrices that are not presented in the sparse matrix are termed 'fill-ins'). In this case, the direct method requires more memory to store data and considerably more time to compute. That why it is very important to study the structure of the sparse matrix to reorder it into a form that requires less 'fill-ins' in the factorization step.

There are a number of ordering techniques available in the literature, for example, the Minimum Degree Ordering (see, for example, (George and Liu (1989), Markowitz (1957), and Tinney and Walker (1967))) and the Nested Dissection Ordering (see, for example, George (1973)). There are also a number of useful reordering library packages. But the reordering by itself is computationally expensive and, as mentioned, in some cases outweighs the factorization step (Hu and Scott, 2005). Therefore, reordering the matrix with prior knowledge of its structure is preferable to using automated reordering packages.

One possible way to reorder the sparse matrix for fast parallel solving is to reorder the matrix in block diagonal form. Duff and Scott (2004) proved that the sparse matrix problem can be effectively solved in parallel by reordering it into SBBD form.

Ordering a matrix into SBBD form is an ordering technique that reorders the row and column of a matrix into the following form:

$$\begin{pmatrix} A_1 & & & & C_1 \\ & A_2 & & & C_2 \\ & & \cdots & & \cdots \\ & & & \cdots & \cdots \\ & & & A_K & C_K \end{pmatrix}$$

where A_i ($i \in [1..K]$) is a rectangular n_i by m_i ($n_i \geq m_i$) matrix and C_i is n_i by L matrix. A_i are block diagonal matrices and C_i are border matrices. K is the number of blocks in the matrix and the number of column in C_i (i.e., L) is called a 'netcut'. We consider only the case of square SBBD matrix, hence $\sum_{i=1}^{K} n_i = \sum_{i=1}^{K} m_i + L$.

The linear equation system will have the form:

$$\begin{pmatrix} A_1 & & & C_1 \\ & A_2 & & C_2 \\ & & \cdots & \cdots \\ & & A_K & C_K \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ . \\ x_K \\ x_L \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ . \\ b_K \end{pmatrix} \tag{4.1}$$

where x_i ($i \in [1..K]$) are the vectors with length m_i, x_L has the length of L, and b_i are the vectors with length n_i.

According to Duff and Scott (2004),[1] solving a linear SBBD system involves factorisation, with forward and backward substitutions. Many parts of the process can be done independently in parallel, using the following steps:

1) The factorisation: we first look at the sub-system

$$(A_i \quad C_i)\begin{pmatrix} x_i \\ x_L \end{pmatrix} = b_i \tag{4.2}$$

where each of the sub-matrices (A_iC_i) will be partially decomposed into the form:

$$(\mathrm{A_i\ C_i}) = P_i \begin{pmatrix} L_i & \\ \tilde{L}_i & I \end{pmatrix} \begin{pmatrix} U_i & \tilde{U}_i \\ & S_i \end{pmatrix} Q_i \tag{4.3}$$

and where P_i, Q_i are row and column permutation matrices, L_i and U_i are a lower and upper $m_i \times m_i$ square matrix, $\tilde{L}_i$ is $(n_i - m_i) \times m_i$ is a rectangular matrix, I is $(n_i - m_i) \times (n_i - m_i)$ an identity matrix, $\tilde{U}_i$ is $m_i \times L$ a rectangular matrix, and S_i is $(n_i - m_i) \times L$ local Schur complement matrix.[2] The idea is to separate the block equation system into two independent parts and solve them separately. Nevertheless, because the local Schur complement matrix is not square (the number of variables x_L is greater than the number of equations $n_i - m_i$), we cannot solve for the partial solution immediately. Fortunately, all partial Schur complement matrices (S_i) can be combined together into a square $L \times L$ interface matrix as we shall see in the next steps.

2) Forward elimination: with the above factorisation, in this second step, we will use the factors to solve for a partial solution $\begin{pmatrix} y_i \\ \tilde{y}_i \end{pmatrix}$:

$$P_i \begin{pmatrix} L_i & \\ \tilde{L}_i & I \end{pmatrix} \begin{pmatrix} y_i \\ \tilde{y}_i \end{pmatrix} = \begin{pmatrix} \hat{b}_i \\ \tilde{b}_i \end{pmatrix} \tag{4.4}$$

where y_i and $\hat{b}_i$ are the vectors of length m_i and $\tilde{y}_i$ and $\tilde{b}_i$ are vectors of length $n_i - m_i$. The above factorisation and forward elimination steps can be done separately for each block matrices (A_iC_i), and hence we can exploit a parallel computing resource here.

3) Solving the interface problem: before going to the backward substitution step (Eq. (4.9) below), we have:

$$\begin{pmatrix} U_i & \tilde{U}_i \\ & S_i \end{pmatrix} Q_i \begin{pmatrix} x_i \\ x_L \end{pmatrix} = \begin{pmatrix} y_i \\ \tilde{y}_i \end{pmatrix} \tag{4.5}$$

as we noted earlier, and because we know y_i and $\tilde{y}_i$, the backward substitution Eq. (4.5) can be solved in two parts for x_i and x_L separately. Because S_i is rectangular with fewer rows than columns, we can assemble all S_i and $\tilde{y}_i$ together to form a complete interface problem. The above partial solutions $\tilde{y}_i$ together with partial Schur complement matrices can be "summed up" to form a larger problem:

$$Sx_L = \tilde{y}_L \tag{4.6}$$

where S is $L \times L$ matrix.[3]

The interface problem again can also be solved using a LU decomposition of matrix S ($S = P_sL_sU_sQ_s$):

$$P_sL_sz_L = \tilde{y}_L \tag{4.7}$$

$$U_sQ_sx_L = z_L \tag{4.8}$$

because the interface problem will be solved serially, it is very important to keep the magnitude of L (i.e., the netcut) low.

4) Backward substitution: once x_L is known, it can be substituted into Eq. (4.5) to find the rest of the unknowns x_i:

$$\left(U_i\tilde{U}_i\right)Q_i\begin{pmatrix} x_i \\ x_L \end{pmatrix} = y_i. \tag{4.9}$$

Again, the backward substitutions of system i can be done independently of each other, so that, again, parallel computing resources can be employed here to shorten the computational time and memory usage.

All of the above procedures can be done easily with the current 'state of the art' HSL parallel direct solver package HSL_MP48 (HSL, 2013), so the real challenge is to reorder the first order partial derivative matrix of an intertemporal CGE model into SBBD form, which we show in the next section.

5. Intertemporal CGE model and SBBD form

A typical intertemporal CGE model can be represented in the following form[4]:

$$p_t = f(p_t, z_t, \lambda_t, q) \tag{5.1}$$

$$z_t = k(p_t, z_t, \lambda_t, q) \tag{5.2}$$

$$\dot{\lambda}_t = h(p_t, z_t, \lambda_t, q) \tag{5.3}$$

$$q = g(z_t, \lambda_t, q) \tag{5.4}$$

where p_t, z_t are vectors of intra-period variables, λ_t is a vector of intertemporal variables, and q is the vector of those variables which do not have a time index. We call Eqs. (5.1), (5.2) intra-period equations and Eq. (5.3) inter-period equations. Eq. (5.4) is called a 'non-time equation', which will not be repeated over time in the model or can be

[1] The reader interested in further details should consult the original paper. We summarise the solution method here for convenience.

[2] To see why S_i is a Schur complement matrix we use a simple common permutation and partition of the block matrices, or:

$$(A_iC_i) \equiv P_i\begin{pmatrix} A_{ii} & A_{ij} \\ A_{ji} & A_{jj} \end{pmatrix}Q_i \equiv P_i\begin{pmatrix} L_i & \\ A_{ji}U_i^{-1} & I \end{pmatrix}\begin{pmatrix} U_i & L_i^{-1}A_{ij} \\ & A_{jj} - A_{ji}U_i^{-1}L_i^{-1}A_{ij} \end{pmatrix}Q_i$$

Then $\tilde{L}_i = A_{ji}U_i^{-1}$, $\tilde{U}_i = L_i^{-1}A_{ij}$ and $S_i = A_{jj} - A_{ji}U_i^{-1}L_i^{-1}A_{ij}$ is a Schur complement of A_{ii}, by definition.

[3] This is because $\sum_{i=1}^{K}(n_i - m_i) = L$.

[4] We will not consider the case of second or higher differential equations. Higher order differential equations are rare in intertemporal CGE models, and especially for deterministic cases.

explicitly assigned to any time period. An example of a non-time equation is an information equation, which has a sum of a variable (in z_t) over time or designates a variable with 2 different time periods on the right hand side.

Note that p_t and z_t are different because of their involvement in non-time equations. If a variable is a part of a non-time equation then we re-order it to the right hand side of the matrix, as we shall see later.[5]

In the finite difference form, the intertemporal CGE model can be written as follows[6]:

$$p_t = f(p_t, z_t, \lambda_t, q) \tag{5.5}$$

$$z_t = k(p_t, z_t, \lambda_t, q) \tag{5.6}$$

$$\lambda_{t+1} - \lambda_t = h(p_t, z_t, \lambda_t, q) \tag{5.7}$$

$$q = g(z_t, \lambda_t, q) \tag{5.8}$$

or more compactly:

$$0 = f^1(p_t, z_t, \lambda_t, q) \tag{5.9}$$

$$0 = k^1(p_t, z_t, \lambda_t, q) \tag{5.10}$$

$$0 = h^1(p_t, z_t, \lambda_t, \lambda_{t+1}, q) \tag{5.11}$$

$$0 = g^1(z_t, \lambda_t, q). \tag{5.12}$$

The first-order partial derivative matrix is:

$$\begin{pmatrix} f^1_{p_t} & f^1_{z_t} & f^1_{\lambda_t} & f^1_q \\ k^1_{p_t} & k^1_{z_t} & k^1_{\lambda_t} & h^1_q \\ h^1_{p_t} & h^1_{z_t} & h^1_{\lambda_t} & h^1_q \\ 0 & g^1_{z_t} & g^1_{\lambda_t} & g^1_q \end{pmatrix}.$$

It is easy to see that the first column of the first-order partial derivative matrix

$$\begin{pmatrix} f^1_{p_t} \\ k^1_{p_t} \\ h^1_{p_t} \end{pmatrix}$$

can be reordered to form disjoint (intra-period) diagonal blocks (A_i, $i \in [1..K]$ as in Eq. (4.1)) and the columns

$$\begin{pmatrix} f^1_{z_t} & f^1_{\lambda_t} & f^1_q \\ k^1_{z_t} & k^1_{\lambda_t} & h^1_q \\ h^1_{z_t} & h^1_{\lambda_t} & h^1_q \\ g^1_{z_t} & g^1_{\lambda_t} & g^1_q \end{pmatrix}$$

can form a joined border part C_i, $i \in [1..K]$ as in Eq. (4.1). In order to accomplish this, we reorder both variables and equations. With proper ordering steps we can form the vector of unknowns according to the following order: ($p_0,\ldots,p_T$, $z_0,\ldots,z_T$, $\lambda_0,\ldots,\lambda_T$, q), where p_t, z_t and λ_t are vectors of all intra-period and intertemporal variables at $t \in [0..T]$, so that the part of first-order partial derivative matrix corresponding to vector ($z_0,\ldots,z_T$, $\lambda_0,\ldots,\lambda_T$, q) will form the border.[7] With the equations also ordered by time and with non-time equations distributed into the lowest (after final time T) part, the first order matrix will be directly transformed into a SBBD matrix. For completeness, the vector of variables should be divided into two parts: endogenous and exogenous parts (with exogenous variables, the number of equations will be less than the number of variables) and the first-order partial derivative matrix should also be partitioned into two corresponding matrices f_x and f_y as in Eq. (3.2). The task can be done easily by dropping exogenous elements in the variables vector while keeping the order of the remaining endogenous elements. The first-order partial derivative matrix should also be partitioned accordingly. The dropped variables will form a vector of exogenous variables. Their order is not important as long as it is kept in line with the new partitioned f_y matrix, since we will only be interested in $f_y dy$ as in Eq. (3.2).

Note that the reason why the joined border part cannot form a disjointed part in the diagonal is because either they contain both non-zero partial derivatives with respect to λ_t and λ_{t+1}, or a no-time indexed equation cannot be part of the intra-period blocks, which have time subscripts. As stated above, partial derivatives with respect to variables in no-time equations (z_t) also cannot be part of diagonal blocks.

The SBBD matrix can be easily solved by a parallel direct solver package HSL_MP48 (HSL, 2013) as indicated in Section 4. Note, however, that even though our approach and the MSG algorithm are different, intuitively they share much in common. The factorisation and forward elimination steps in the LU decomposition of SBBD matrices (Section 4) are similar to the elimination of endogenous variables in the MSG algorithm, while the interface problem solution is, in a sense, equivalent to iterative solution of condensed difference equations in the MSG algorithm. In both methods, the special feature of intertemporal CGE models has been exploited to solve them faster. By employing parallel computing resources to do the factorisation, along with the forward and backward eliminations, our method will be even more efficient.

6. Numerical analysis

The model we examine here is an regional intertemporal model for the Vietnamese economy (Ha and Kompas, 2009). The model was developed from a single country ORANI model (Dixon et al., 1982). The ORANI model was originally built for the Australian economy, but lately has been applied to many country with its latest version of ORANI-G (Horridge, 2003).

The model can be varied from 3 to 28 commodities and industries, with intra-temporal and intertemporal blocks, and has 3 agents in every region of the country's eight regions: a consumer, a producer and a government. In every region and within each period of time (i.e., an intra-temporal block), the model replicates ORANI-G's structure, where each household and government forms demands as a combination of Constant Elasticity of Substitution (CES) and Stone–Geary functions. On the production side, output is produced from intermediate inputs and factors of production (i.e., capital, labour, and land) in a combination of Leontief and CES production functions. Output is then sold within the region, to other regions or abroad. Household income is obtained from possession of labour and productive factors, while the government's income is from taxes and duties. General equilibrium will be ensured through market clearing conditions: goods produced in one region are equal to the demand for that good in other regions and from abroad, and demand equals supply for all productive factors. An intertemporal block includes three components: (1) producers maximise the value of the firm in the long run subject to a capital accumulation equation; (2) consumers maximise utility in the long run subject to a per capita debt level; and (3) a set of financial equations link money with the price level, and a version of the uncovered interest rate parity condition links exchange rate movements with the difference between domestic and international interest rates.

The model uses an illustrative inter-regional Input–Output database of Vietnamese economy. The database has been described and

[5] This is mostly for simplicity in coding. We find non-time equations rare, and we can order any variable to the right hand side of the matrix (classified as z_t) with the same solution algorithm, noting that the interface problem will be larger.

[6] Eq. (5.7) can take the form of a forward or backward difference equation depending on whether λ is a stock or jump variables. We write the equation in this form for simplicity.

[7] Because there is no assumption about the shape of the border part C_i, ordering (z0,…,zT, λ0,…,λT, q) is not necessary.

constructed for a more elaborate intertemporal CGE model in Ha and Kompas (2009) for the year 2005 and updated to the year 2007 in [see,] Thang et al. (2011), for a description of the updated database.

To test the performance of the solution methods, we solve the above model with different aggregate levels and different finite difference grids. We increase the size of model from 3 commodities by 3 industries to 28 commodities by 28 industries. At the same time we also vary the number of discretization intervals from time zero to the steady state from 10 to 50. Because we represent differential equations in discretization from, the higher the number of intervals the more precise is the solution of the model. The same model will thus expand from a few hundred thousand variables into a huge system with more than 32.7 million endogenous variables (see Table 1).

The numerical test is done using a software package developed at the Australian Centre for Biosecurity & Environmental Economics, at the ANU Crawford School of Public Policy. The package is designed specifically to solve large intertemporal CGE models in a parallel computing environment. The package relies on HSL's MP48 (HSL, 2013) package to solve a large SBBD matrix in parallel. The parallel matrix manipulation, transformation and representation are carried out using PETSC (Balay et al., 1997, 2013, 2014). All computational experiments in the paper has been carried out on a MacBook Air 2012 model with dual processors and 8 gigabyte of RAM as the leading computer and 3 other slave computers (see Appendix A for details).

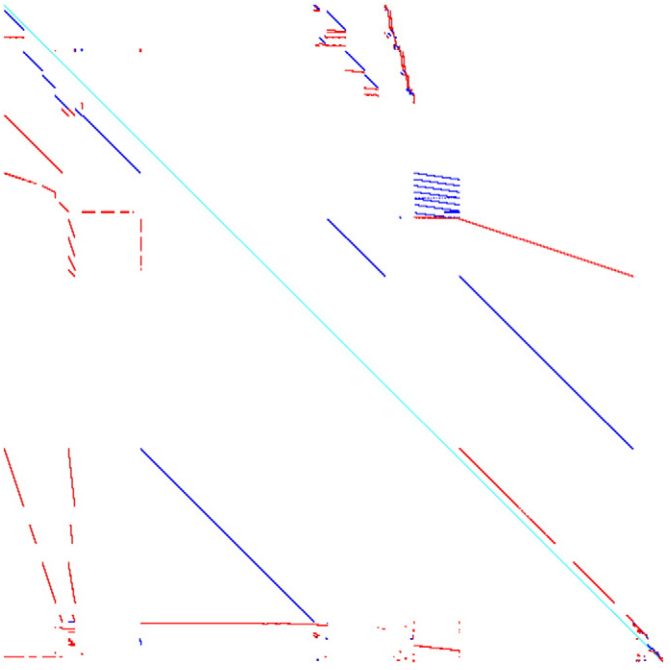

Fig. 1. The matrix without ordering. Source: Author's calculation.

6.1. Graphical representation of the direct ordering technique

Fig. 1 represents the "un-ordered" matrix with a conventional variables-by-variables and equations-by-equations ordering method. The figure is a graphical representation of the first-order partial derivative matrix f_x (as defined in Eq. (3.2)) with "un-ordered" variables and equations. Red, blue and cyan represent positive values, negative values and zero elements on the matrix diagonal. Off diagonal zero elements are not coloured. The matrix drawing function is from PETSC (Balay et al., 1997, 2013, 2014). According to the figure, the matrix is clearly unstructured and will require ordering techniques before any factorization stage can be carried out. Fig. 2, alternatively, represents the matrix re-ordered by our proposed technique. Since we use prior knowledge to reorder the matrix, the reordered matrix is very stable (i.e., a low row difference between blocks) and has a small netcut.

6.2. Comparing the ordering techniques

To test the numerical performance of the direct ordering method, we compare it against a conventional reordering package HSL_MC66 (HSL, 2013). With small models, HSL_MC66 performed well in having a smaller netcut compared to our direct method. That said, the MC66 still can never dominate over our direct method in row difference because our intertemporal model has nearly an identical structure over time and the direct method exploits this fact. In any case, when the model grows in size, the automatic reordering method starts to quickly lag behind our direct method. More importantly, as we have discussed earlier, the ordering package requires substantial memory to perform its operation, and since it is serial, when the model size passes 26 millions variables it fails to allocate enough memory to even reorder the matrix. Our direct method, on the other hand, does not require much more memory than that needed to store the matrix itself, hence we can manage to reorder a 32.7 million by 32.7 million sparse matrix to pass to the factorisation step (see Table 2).

6.3. The size of intertemporal CGE models and serial computing time

The ordering methods are very important for the matrix LU decomposition, and the LU decomposition should begin with a prediction of future elements in the LU factor matrices (the 'fill-ins'). With proper ordering techniques, the number of 'fill-ins' could be minimised and therefore the efficiency of the decomposition method will be enhanced. Keeping this in mind, our first experiment is to compare the serial solution methods to show the efficiency of the ordering techniques.

We will compare the numerical performance of the direct method reordering with the HSL_MC66 package and also compare its performance with the conventional serial LU decomposition method MA48. The matrix solver packages are HSL_MP48 and MA48 from the HSL library (HSL, 2013).

Results from Table 3 show the clear advantage of our direct ordering method, even though no parallel resources were employed. The direct method + HSL_MP48 out-performs HSL_MC66 + HSL_MP48 in every model. Especially for larger models, when the model's size exceeds 26 million variables (Model ID No. 4), HSL_MC66 + HSL_MP48 failed (due to insufficient memory in ordering stage), while the combination of direct ordering method and HSL_MP48 solver solves the model in just 8.7 min.

When we compare the direct ordering method with the conventional serial MA48 solution method (without prior re-ordering), the efficiency of our method is also striking. The MA48 method outperforms

Table 1
Model descriptions.

ID	Model size	Number of endogenous variables	Number of exogenous variables	Number of non-zeros
1	3 sectors, 3 commodities, 8 regions and 10 time periods	246,833	55,467	779,466
2	3 sectors, 3 commodities, 8 regions and 50 time periods	1,144,433	257,067	3,614,846
3	8 sectors, 8 commodities, 8 regions and 50 time periods	5,720,368	1,298,272	16,806,197
4	28 sectors, 28 commodities, 8 regions and 20 time periods	26,422,686	7,944,170	70,680,341
5	28 sectors, 28 commodities, 8 regions and 25 time periods	32,713,851	9,841,409	87,392,893

Source: Author's calculation.

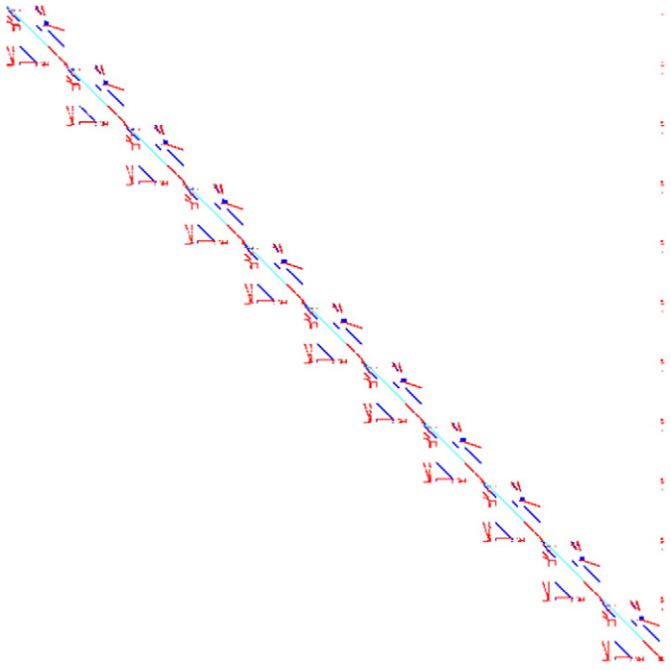

Fig. 2. The reordered matrix. Source: Author's calculation.

our method with small models (Model 1) due to overhead computational cost. With lager models, however, our method clearly performs better, with computational time reduced by nearly 6 times in the case of Model 5, or 11.2 min computational time in comparison with more than an hour.

6.4. Parallel computing performance

The advantage of the direct ordering method is even further enhanced when parallel computing resources are employed. To test the performance of the parallel solution we solve models 1 to 5 mobilising 4–10 processes in 1 to 4 machines over a network (see columns 2–5 in Table 4). The machines are typical working machines which can be found in nearly any office (3 Dell computers and a MacBook Air, see Appendix A for details). The performance of the parallel computing platform is clearly far better than that with serial computing (column 4, Table 3).

Even using multicores in the same computer, the performance of the parallel solution is better than the serial solution. Column 2 of Table 4 shows that with 3 processes employed, we can cut 40% of the computing time over that of serial solution in the case of the Model 5 (see also column 3 of Table 3).

Given the fact that computers on the network have to connect with each other by cable and through a network switch, the network computers do not have an advantage over machines with the same core.

Table 2
Matrix ordering.

ID	No. of blocks	Netcut		Row difference[†]	
		HSL MC66	Direct reordering	HSL MC66	Direct reordering
1	11	595	719	0.01%	0.00%
2	51	3011	3359	1.01%	0.00%
3	51	26,563	7399	62.94%	0.00%
4	21	Failed	9579	Failed	0.00%
5	26	Failed	11,909	Failed	0.00%

Source: Author's calculation.
[†] Row difference is $(m_{max} - N/B)/(N/B) \times 100$, where m_{max} is the largest number of row in a block, N is the total number of rows, and B is the number of blocks (see (Hu and Scott, 2005)).

Table 3
Calculation time (in sec.).

ID	HSL_MC66 + HSL_MP48	Direct reordering + HSL_MP48	MA48
1	1.130868	0.536218	0.296910
2	7.569205	2.580329	4.385017
3	58.688180	28.676437	126.969160
4	Failed	523.388481	1790.043886
5	Failed	673.380096	3786.980331

Source: Author's calculation.
Notes: (1) All numerical experiments are carried out with a MacBook Air (see Appendix A). (2) All numerical experiments are carried out with a Johansen one step simulation (see for example (Pearson (1991) and Dixon et al. (1992))). (3) Time is counted for the linear system (matrix) solution only.

However, this disadvantage will be offset when more processes are being employed. When 4 processes are mobilised, the computing time for Model 5 is cut further down by half (in comparison with the serial computation time), and the computing time reaches its best performance when 10 processes are employed (see columns 3–5 of Table 4). Computation time is now 4–5 min to solve Model 5 in comparison with more than an hour using the conventional serial solution method without our direct reordering method. This is a significant improvement. Computational efficiency will even be further enhanced in a (GEMPACK style) multi-step solution, which is needed if we want a more accurate solution.

In general, and as discussed earlier, the HSL_MP48 package will solve each of the diagonal and border blocks separately, followed by the interface matrix, and a 'backward solving' for the rest of unknowns. The size of interface matrix will decide the efficiency of the method. Given the stable structure of our intertemporal CGE model the interface matrix does not grow as fast as the first-order derivative matrix itself, with its denser finite difference grid. Only variables that enter in the border section will have extra elements when the time grid increases. This, in essence, allows us to solve a much larger model more efficiently in parallel.

6.5. Accuracy of finite difference methods in intertemporal CGE models

Intertemporal CGE models require a solution to both intra- and inter-period equations. The inter-period equations are usually the forward–backward differential equations, for which the finite difference method is always the best choice [see, for example,] Dixon et al. (1992). For the finite difference method, which is the case in our numerical experiment, it is crucial that the finite difference step is as small as possible. When the models satisfy regularity conditions, the numerical solution of the finite difference method will converge to the true solution as the finite difference steps go to zero. We could also vary the size of steps between the initial and end point (in time) to get acceptable accuracy with smaller grid size (see Dixon et al. (1992)).

To get an idea of how step size affects the numerical solution we draw a graph of GDP change in Model 1 with an assumed technological shock under different finite difference grid sizes. The solution method is

Table 4
Parallel computing performance (in sec.).

ID	3 processes on machine 1	4 machines and 4 processes	4 machines and 6 processes	4 machines and 10 processes
1	0.341564	0.455525	0.397198	0.314949
2	1.880380	2.076181	1.693513	1.548978
3	20.477740	16.883149	12.873069	12.470178
4	354.354870	222.596670	207.515911	170.984501
5	407.100470	324.550223	283.381909	274.561792

Source: Author's calculation.
(1) Numerical experiments are carried out with a MacBook Air and 3 Dell computers (see Appendix A). (2) All numerical experiments are carried out with a Johansen one step simulation (see for example (Pearson (1991) and Dixon et al. (1992))). (3) Time is counted for the linear system (matrix) solution only.

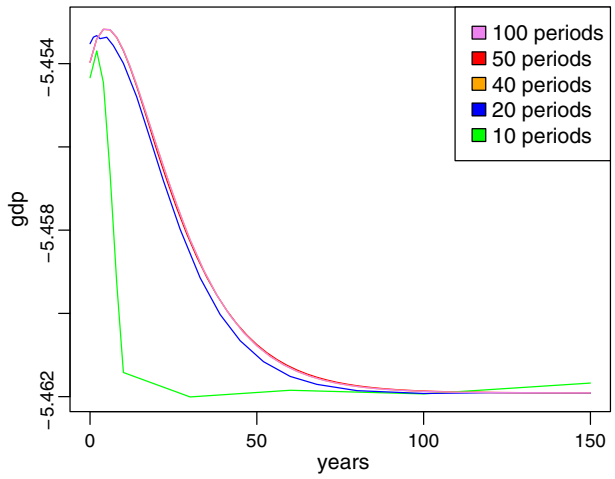

Fig. 3. The finite difference grid and calculation accuracy (% change in GDP). Source: Author's calculation.

Gragg with 2–4–8 steps (Dixon et al., 1992; Pearson, 1991) and the time horizon is 150 years. Although with a proper variation in grid steps (i.e., higher density in initial periods), the errors between different grid sizes can be minimised, the denser grid usually provides better results. Fig. 3 shows a clearly inaccurate solution when the number of periods (grid size) is limited to 10. The solution, in other words, is not smooth when it is supposed to be, that is, when agents in the economy are smoothing out investment and consumption. For more than 20 periods solutions tend be likely solution candidates with GDP levels gradually and smoothly increasing toward the steady state over time.

However, the more periods we include in the numerical solution, the larger the intertemporal model will be. In our case, with only 3 sectors and 3 commodities, the number of endogenous variable increases from as little as 250 thousand endogenous variables (and equations) to more than a million variables, while the grid size increases from 10 to 50 periods. Clearly, parallel solutions will be essential to achieve efficiency in computational time.

7. Concluding remark

From our numerical experiments, the direct reordering method of the first-order derivative matrix of intertemporal CGE models into SBBD form is shown to be very efficient in solving intertemporal CGE models in parallel. The saving in computational time and memory requirements is substantial, allowing researchers to solve larger scale CGE models more quickly and accurately. The efficiency of the method will vary depending on the size of the interface problem. Researchers can try to substitute out q to reduce this problem. Nevertheless, with the rest of the interface matrix depending only on inter-period variables λ_t, it size will increase moderately (in comparison with the entire model's size) as the finite grid size increases. Our method thus proves to be the best and most practical approach to solving intertemporal CGE models in parallel.

Acknowledgements

The computations in this paper are performed with the help of: Portable, Extensible Toolkit for Scientific Computation (PETSc) at Argonne National Laboratory (Balay et al., 1997, 2013, 2014); Message Passing Interface (MPICH); HSL Mathematical Software Library (HSL, 2013); and the GNU Compiler Collection (GCC). A detailed list of software can be found in Appendix B. We would like to thank the developers of the software for making their work available.

Appendix A. Hardware configuration

The parallel computing exercises are carried out with 4 computers:

1. MacBook Air 2012 with Intel® Core™ i7-3667U CPU @ 2.00 GHz × 4, 500 GB SSD.
2. Dell Optiplex 745 with Intel® Core™ 2 CPU (6400@2.13× GHz), 2 GB of RAM, 200 GB Hard disk.
3. Dell Optiplex 755 with Intel® Core™ 2 (E6550@2.33× GHz), 2 GB of RAM, 200 GB Hard disk.
4. Dell Optiplex 755 with Intel® Core™ 2 Quad (Q6600@2.4 GHz × 4), 4 GB of RAM, 200 GB Hard disk.

The network device is a TP-LINK TL-SG1008D Gigabit switch.

Appendix B. Software

Table B.5
The list of software and libraries used in the numerical simulation.

	Name	Version	Source
Operating system	Ubuntu	14.04	http:www.ubuntu.com
Compiler	GCC	4.8.2	http:gcc.gnu.org
Message passing interface	MPICH2	3.0.4	http:www.mpich.org
Matrix manipulation	Petsc	3.4.4	http:www.mcs.anl.govpetsc
Matrix solver	HSL[1]	2013	www.hsl.rl.ac.uk

Note: [1]Subpackages used: HSL_MP48, MA48,HSL_MC66.

References

Anstreicher, K., Lee, J., Rutherford, T., 1992. Crashing a maximum-weight complementary basis. Math. Program. 54, 281–294. http://dx.doi.org/10.1007/BF01586055.
Balay, S., Gropp, W.D., McInnes, L.C., Smith, B.F., 1997. Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A.M., Langtangen, H.P. (Eds.), Modern Software Tools in Scientific Computing. Birkhäuser Press, pp. 163–202.
Balay, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Rupp, K., Smith, B.F., Zhang, H., 2013. PETSc Users Manual. Technical Report ANL-95/11 — Revision 3.4. Argonne National Laboratory.
Balay, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Rupp, K., Smith, B.F., Zhang, H., 2014. PETSc Web page. http://www.mcs.anl.gov/petsc.
Billups, S., Dirkse, S., Ferris, M., 1997. A comparison of large scale mixed complementarity problem solvers. Comput. Optim. Appl. 7, 3–25. http://dx.doi.org/10.1023/A:1008632215341.
Devarajan, S., Robinson, S., 2013. Chapter 5 — contribution of computable general equilibrium modeling to policy formulation in developing countries. In: Dixon, P.B., Jorgenson, D.W. (Eds.), Handbook of Computable General Equilibrium Modeling SET, Vols. 1A and 1BHandbook of Computable General Equilibrium Modeling vol. 1. Elsevier, pp. 277–301. http://dx.doi.org/10.1016/B978-0-444-59568-3.00005-5.
Dirkse, S.P., Ferris, M.C., 1995. The PATH solver: a non-monotone stabilization scheme for mixed complementarity problems. Optim. Methods Softw. 5, 123–156.
Dixon, P.B., Jorgenson, D.W., 2013. Chapter 1 — Introduction. In: Dixon, P.B., Jorgenson, D.W. (Eds.), Handbook of Computable General Equilibrium Modeling SET, Vols. 1A and 1BHandbook of Computable General Equilibrium Modeling vol.1. Elsevier, pp. 1–22. http://dx.doi.org/10.1016/B978-0-444-59568-3.00001-8.
Dixon, P.B., Parmenter, B., 1996. Chapter 1: Computable general equilibrium modelling for policy analysis and forecasting. In: Hans, M., Amman, D.A.K., Rust, J. (Eds.), Handbook of Computational Economics vol. 1. Elsevier, pp. 3–85. http://dx.doi.org/10.1016/S1574-0021(96)01003-9.
Dixon, P.B., Rimmer, M.T., 2002. Dynamic General Equilibrium Modelling for Forecasting and Policy: A Practical Guide and Documentation of MONASH. Contributions to Economic Analysis vol. 256. North-Holland Publishing Company, Amsterdam.
Dixon, P.B., Rimmer, M.T., 2004. The US economy from 1992 to 1998: results from a detailed CGE model. Econ. Rec. 80, S13–S23. http://dx.doi.org/10.1111/j.1475-4932.2004.00180.x.
Dixon, P., Parmenter, B., Sutton, J., Vincent, D., 1982. ORANI: A Multisectoral Model of the Australian Economy. Contributions to Economic Analysis 142. North-Holland Publishing Company.
Dixon, P.B., Parmenter, B., Powell, A.A., Wilcoxen, P.J., 1992. Notes and problems in applied general equilibrium economics. In: Bliss, C., Intriligator, M. (Eds.), Advanced Textbooks in Economics vol. 32. North-Holland Publishing Company, Amsterdam, London, New York, Tokyo, p. xvi +392.
Duff, I.S., Scott, J.A., 2004. A parallel direct solver for large sparse highly unsymmetric linear systems. ACM Trans. Math. Softw. 30, 95–117. http://dx.doi.org/10.1145/992200.992201.
EC, I.M.F., OECD, U.N., Bank, World, 2009. System of National Accounts 2008 (New York).

George, A., 1973. Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. 10, 345–363. http://dx.doi.org/10.1137/0710032, % tt arXiv (http://epubs.siam.org/doi/pdf/10.1137/0710032).
George, A., Liu, W.H., 1989. The evolution of the minimum degree ordering algorithm. SIAM Rev. 31, 1–19. http://dx.doi.org/10.1137/1031001.
Gill, P.E., Murray, W., Saunders, M.A., Wright, M.H., 1987. Maintaining LU Factors of a General Sparse Matrix. Linear Algebra and its Applications 88–89pp. 239–270 http://dx.doi.org/10.1016/0024-3795(87)90112-1.
Ha, P.V., Kompas, T., 2009. Xay dung mo hinh can bang tong the dong lien vung cho nen kinh te Viet Nam. Chuyen de nghien cuu cap Vien, Vien Khoa hoc Tai chinh. Hoc vien Tai chinh, Bo Tai chinh.
Harrison, W., Pearson, K., 1996. Computing solutions for large general equilibrium models using GEMPACK. Comput. Econ. 9, 83–127. http://dx.doi.org/10.1007/BF00123638.
Horridge, M., 2003. ORANI-G: A Generic Single-Country Computable General Equilibrium Model. Edition Prepared for the Practical GE Modelling Course June 23–27, 2003. Centre of Policy Studies and Impact Project, Monash University, Australia.
Horridge, J.M., Pearson, K., 2006. Running simulations faster on multi-processor or 64-bit PCs via GEMPACK. CoPS/IMPACT Working Paper Number C15–01.
Horridge, M., Meeraus, A., Pearson, K., Rutherford, T.F., 2013. Chapter 20 — Solution software for computable general equilibrium modeling. In: Dixon, P.B., Jorgenson, D.W. (Eds.), Handbook of Computable General Equilibrium Modeling SET, Vols. 1A and 1BHandbook of Computable General Equilibrium Modeling vol. 1. Elsevier, pp. 1331–1381. http://dx.doi.org/10.1016/B978-0-444-59568-3.00020-1.
HSL, 2013. A collection of fortran codes for large scale scientific computation. http://www.hsl.rl.ac.uk.
Hu, Y., Scott, J., 2005. Ordering techniques for singly bordered block diagonal forms for unsymmetric parallel sparse direct solvers. Numer. Linear Algebra Appl. 12, 877–894. http://dx.doi.org/10.1002/nla.427.
Johansen, L., 1960. A Multi-Sector Study of Economic Growth. North-Holland Pub. Co.
Julia-Wise, R., Cooke, S.C., Holland, R., 2002. A computable general equilibrium analysis of a property tax limitation initiative in Idaho. Land Econ. 78, 207–227.
Leontief, W.W., 1936. Quantitative input and output relations in the economic systems of the United States. Rev. Econ. Stat. 18, 105–125.
Malakellis, M., 1998. Should tariff reductions be announced? An intertemporal computable general equilibrium analysis. Econ. Rec. 74, 121–138.
Markowitz, H.M., 1957. The elimination form of the inverse and its application to linear programming. Manag. Sci. 3, 255–269.
McKibbin, W.J., 1987. Numerical Solution of Rational Expectations Models With and Without Strategic Behaviour. RBA Research Discussion Papers rdp8706. Reserve Bank of Australia.
McKibbin, W.J., Sachs, J.D., 1991. Global Linkages: Macroeconomic Interdependence and Cooperation in the World Economy. Brookings Institution, Washington D.C.
McKibbin, W.J., Wilcoxen, P.J., 1999. The theoretical and empirical structure of the G-Cubed model. Econ. Model. 16, 123–148.
Murtagh, B., Saunders, M., 1982. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. In: Buckley, A., Goffin, J.L. (Eds.), Algorithms for Constrained Minimization of Smooth Nonlinear Functions. Springer, Berlin Heidelberg, pp. 84–117 http://dx.doi.org/10.1007/BFb0120949 (volume 16 of Mathematical Programming Studies).
Pearson, K., 1991. Solving nonlinear economic models accurately via a linear representation. Centre of Policy Studies/IMPACT Centre Working Papers ip-55. Centre of Policy Studies/IMPACT Centre, Monash University.
Richardson, L.F., 1911. The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam. Philos. Trans. R. Soc. Lond. Ser. A 210 (459-470), 307–357.
Saad, Y., 2003. Method related to normal equation. Iterative Methods for Sparse Linear Systems, 2nd ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 245–260 (Chapter 8).
Scarf, H., 1967. The approximation of fixed points of a continuous mapping. SIAM J. Appl. Math. 15, 1328–1343.
Scarf, H., Hansen, T., 1973. The Computation of Economic Equilibria. Yale University Press, New Haven and London.
Scollay, R., Gilbert, J., 2000. Measuring the gains from APEC trade liberalisation: an overview of CGE assessments. World Econ. 23, 175–197. http://dx.doi.org/10.1111/1467-9701.00267.
Shoven, J.B., Whalley, J., 1972. A general equilibrium calculation of the effects of differential taxation of income from capital in the U.S. J. Public Econ. 1, 281–321. http://dx.doi.org/10.1016/0047-2727(72)90009-6.
Shoven, J.B., Whalley, J., 1973. General equilibrium with taxes: a computational procedure and an existence proof. Rev. Econ. Stud. 40, 475–489.
Thang, V.N., Ha, P.V., Lien, L.T.M., Thuan, L.Q., Anh, D.P., Van, L.T.T., Thao, N.T.T., Linh, H.D., Son, D.T.T., 2011. Danh gia tac dong ve tai chinh va nhung thach thuc chinh sach dat ra cua van de bien doi khi hau o Viet nam. De tai Nghien cuu khoa hoc cap Bo. Vien Chien luoc va Chinh sach Tai chinh, Bo Tai chinh, Vietnam.
Tinney, W.F., Walker, J., 1967. Direct solutions of sparse network equations by optimally ordered triangular factorization. Proc. IEEE 55, 1801–1809. http://dx.doi.org/10.1109/PROC.1967.6011.
United Nations, 1953. A system of National Accounts and Supporting Tables. Studies in Methods, Series F No 2 (New York).