PARALLEL COMPUTATION*

Anna Nagurney

School of Management

University of Massachusetts

Amherst, Massachusetts 01003

December, 1993

* **Chapter Intended for Handbook of Computational Economics,**

Hans Amman, David Kendrick, and John Rust, editors

1. Introduction

The emergence of computation as a basic scientific methodology in economics has given access to solutions of fundamental problems that pure analysis, observation, or experimentation could not have achieved. Virtually every area of economics from econometrics to microeconomics and macroeconomics has been influenced by advances in computational methodologies. Indeed, one cannot envision the solution of large-scale estimation problems and general economic equilibrium problems without the essential tool of computing.

The ascent of economics to a computational science discipline has been fairly recent, preceded by the earlier arrivals of physics, chemistry, engineering, and biology. Economics, as the other computational disciplines, stands on the foundations established by computer science, numerical analysis, and mathematical programming; unlike the aforementioned computational disciplines, it is grounded in human behavior. As to be expected, it brings along unique computational challenges, which have stimulated research into numerical methods.

For example, many algorithms for the solution of optimization problems, equilibrium problems, including game theory problems, and dynamical systems, among other problem-types, can trace the requirement of the solution of a prototypical economic problem as the motivation for their discovery and subsequent development. Indeed, one has only to recall the early contributions to computational economics of Koopmans (1951), Dantzig (1951), Arrow, Hurwicz, and Uzawa (1958), Dorfman, Samuelson, and Solow (1958), and Kantorovich (1959) in the form of the formulation and solution of linear programming problems derived from resource allocation and pricing problems.

Subsequently, the need to formulate and solve portfolio optimization problems in financial economics, where the objective function is no longer linear and represents risk to be minimized, helped to stimulate the area of quadratic programming (cf. Markowitz (1952, 1959) and Sharpe (1970)). Quadratic programming is also used for the computation of certain spatial price equilibrium problems in agricultural and energy economics (cf. Takayama and Judge (1964, 1971)). The special structure of these problems, as those encountered in portfolio optimization, has further given rise to the development of special-purpose algorithms for their computation (cf. Dafermos and Nagurney (1989)). Quadratic programming has, in addition, provided a powerful tool in econometrics in the case of

certain problems such as the estimation of input/output tables, social accounting matrices, and financial flow of funds tables (cf. Bacharach (1970), Nagurney and Eydeland (1992), Hughes and Nagurney (1992), and the references therein). In such problems, the different values of the quadratic form depict different distributions. Econometrics has also made use of the techniques of global optimization, which is appealed to in the case that the function to be minimized is no longer convex (cf. Pardalos and Rosen (1987), Goffe, Ferrier, and Rogers (1993))).

Nonlinear programming, which contains quadratic programming as a special case, has found wide application in the neoclassical theory of firms and households in microeconomics. In such problems the firms or households seek to maximize a certain objective function subject to constraints (see, e.g., Intriligator (1971), Takayama (1974), and Dixon, Bowles, and Kendrick (1980)).

Recently, there has been an increasing emphasis on the development and application of methodologies for problems which optimization approaches alone cannot address. For example, fixed point algorithms pioneered by Scarf (1964, 1973), complementarity algorithms (cf. Lemke (1980)), homotopy methods, along with the global Newton method and its many variants (cf. Smale (1976), Garcia and Zangwill (1981), and the references therein), and variational inequality algorithms (cf. Dafermos (1983) and Nagurney (1993)) have yielded solutions to a variety of equilibrium problems in which many agents compete and each one seeks to solve his/her own optimization problem. Such classical examples as imperfectly competitive oligopolistic market equilibrium problems, in which firms are involved in the production of a homogeneous commodity, and seek to determine their profit-maximizing production patterns, until no firm can improve upon its profits by unilateral action, and Walrasian price equilibrium problems in which agents, given their initial endowments of goods, seek to maximize their utilities, by buying and selling goods, thereby yielding the equilibrium prices and quantities, fall into this framework.

The drive to extend economic models to dynamic dimensions, as well as the desire to better understand the underlying behavior that may lead to an equilibrium state, led to the introduction of classical dynamical systems methodology (cf. Coddington and Levinson (1955), Hartman (1964), Varian (1981)). Recently, it has been shown in Dupuis and Nagurney (1993) that the set of stationary solutions of a particular dynamical system cor-

responds to the set of solutions of a variational inequality problem. The dynamical system has its own application-specific constraints, and is non-classical in that its right-hand side is discontinuous. Such a dynamical system, for example, could guarantee that prices are always nonnegative as well as the commodity shipments. Moreover, this dynamical system can be solved by a general iterative scheme. This scheme contains such classical methods as the Euler method, the Heun method, and the Runge-Kutta method as special cases.

Finally, the need to incorporate and evaluate the influence of alternative policies in economic models in the form of dynamical systems has yielded innovations in stochastic control and dynamic programming (cf. Kendrick (1973), Holbrook (1974), Chow (1975), Norman (1976), Judd (1971)). In contrast to the focus in dynamical systems, where one is interested in tracing the trajectory to a steady state solution, in control theory, the focus is on the path from the present state to an improved state. Moreover, unlike the aforementioned mathematical programming problems, where the feasible set is a subset of a finite-dimensional space, the feasible set is now infinite-dimensional (see also Intriligator (1971)).

Computational methodologies, hence, have greatly expanded the scope and complexity of economic models that can now be not only formulated, but analyzed and solved. At the same time, the increasing availability of data is pushing forward the demand for faster computer processors and for greater computer storage, as well as for even more general models with accompanying new algorithmic approaches. Faster computers also enable the timely evaluation of alternative policy interventions, thereby, decreasing the time-scales for analysis, and minimizing the potential costs of implementation. It is expected that algorithmic innovations in conjunction with computer hardware innovations will further push the frontiers of computational economics.

To date, the emphasis in economics has been on serial computation. In this computational framework an algorithm or simulation methodology is implemented on a serial computer and all operations are performed in a definite, well-defined order. The emphasis on serial computation is due to several factors. First and foremost, serial computers have been available much longer than, for example, parallel computers, and, hence, users have not only a greater familiarity with them, but, also, more software has been developed for such architectures. Secondly, many computer languages are serial in nature and,

consequently, the use of such programming languages subsumes basically serial algorithms and their subsequent implementation on serial architectures. Moreover, humans naturally think sequentially although the brain also processes information in parallel. In addition, the use of parallel architectures requires learning not only, perhaps, other programming languages, but also new computer architectures. Finally, the development of entirely new algorithms, which exploit the features of the architectures may be required.

Computation, however, is evolving along with the technological advances in hardware and algorithms, with an increasing focus on the size of the computational problems that can be handled within acceptable time and cost constraints. In particular, parallel computation represents an approach to computation that can improve system performance dramatically, as measured by the size of problem that can be handled. In contrast to serial computation, with parallel computation, many operations are performed simultaneously. Parallel processors may be relatively sophisticated and few in number or relatively simple and in the thousands.

Parallel computation achieves its faster performance through the decomposition of a problem into smaller subproblems, each of which is allocated and solved simultaneously on a distinct processor. For example, in the case of a multinational or multiregional trade problem, the level of decomposition may be "coarse," with the decomposition being on the level of the number of nations or regions, or "fine," as on the level of the commodity trade patterns themselves. In fact, for a given problem there may be several alternative decompositions that may be possible, say on the level of the number of commodities, on the number of markets, or on the number of market pairs.

The technology represented by massively parallel computation, in particular, suggests that there is no obvious upper limit on the computational power of machines that can be built. Two basic concepts emerge here – that of "data parallelism" and that of "scalability." Data parallelism is a technique of coordinating parallel activities, with similar operations being performed on many elements at the same time, and exploits parallelism in proportion to the amount of data (cf. Hillis (1992)). Data level parallelism is to be contrasted with one of the simplest and earliest techniques of coordinating parallel activities, known as pipelining, which is analogous to an assembly line operation, where operations are scheduled sequentially and balanced so as to take about the same amount of time. Vector

processors work according to this principle. Another technique of parallelism that can be used along with pipelining is known as functional parallelism. An example of this in a computer would occur when separate multiplication and addition units would operate simultaneously. Although both of these techniques are useful, they are limited in the degree of parallelism that they can achieve since they are not "scalable".

Scalability envisions building massively parallel computers out of the same components used in, for example, desktop computers. Consequently, the user would be able to scale up the number of processors as demanded by the particular application without a change in the software environment.

Parallel computation represents the wave of the future. It is now considered to be cheaper and faster than serial computing and the only approach to faster computation currently foreseeable (cf. Deng, Glimm, and Sharp (1992)).

Parallel computation is appealing, hence, for the economies of scale that are possible, for the potentially faster solution of large-scale problems, and also for the possibilities that it presents for imitating adjustment or tatonnement processes. For example, serial computation is naturally associated with centralization in an organization whereas parallel (and the allied distributed) computation is associated with decentralization, in which units work, for the most part independently, with their activities being monitored and, perhaps, synchronized, periodically. Market structures associated with central planning are thus more in congruence with serial and centralized computation, with competitive market structures functioning more as parallel systems. Indeed, one can envision the simulation of an economy on a massively parallel architecture with each competing agent functioning as a distinct processor in the architecture, with price exchanges serving as messages or signals to the other agents. The agents would then adjust their behavior accordingly.

To fix some ideas, we now mention several basic issues of parallel architectures, which are discussed further in Section 2. The principal issues are: 1). the type of processing involved, typically the combination of instruction and data parallelism, 2). the type of memory, either global (or shared) or local (or distributed), 3). the type of interconnection network used, and 4). the processing power itself. The interconnection network, for example, is not an issue in a serial architecture. It consists of links joining pairs of processors and provides routes by which processors can exchange messages with other processors, or make

requests to read from, write to, or lock memory locations. In designing interconnection networks for parallel processing systems, every effort is made to minimize the potential for bottlenecks due to congestion on the network. The basic distinction that can be made between processing power is whether or not the processors are simple or complex. The latter are more common in "coarse-grained" architectures, typically consisting of several processors, say on the order of ten, whereas the former – in "fine-grained" architectures, typically consisting of thousands of processors.

The design of a parallel algorithm, hence, may be intimately related to the particular architecture on which it is expected to be used. For example, the parallelization of an existing serial algorithm, a typical, first-cut approach, may fail to exploit any parallel features and realize only marginal (if any) speedups. On the other hand, certain "serial" algorithms, such as simulation methodologies may be, in fact, embarrassingly parallel in that the scenarios themselves can be studied almost entirely independently (and on different processors) with the results summarized at the completion of the simulation. These two situations represent extremes with the most likely situation being that a new and easier to parallelize algorithm will be developed for a particular problem class.

The last decade has revealed that parallel computation is now practical. The questions that remain to be answered are still many. What are the best architectures for given classes of problems? How can a problem be decomposed into several, or into numerous subproblems for solution on, respectively, coarse-grained or fine-grained architectures? How do we design an algorithm so that the load across processors is balanced, that is, all processors are kept sufficiently busy and not idle? What new directions of research in numerical methods will be unveiled by the increasing availability of highly parallel architectures? What new applications to economics remain to be discovered?

Parallel computation represents not only a new mode of computation, but a new intellectual paradigm. It requires one to view problems from a new perspective and to examine problems as comprised of smaller, interacting components, and at different levels of disaggregation. It breaks down barriers between disciplines through a common language. With expected dramatic shifts from serial to parallel computation in the future, a significant change in scientific culture is also envisaged.

In this chapter the focus will be on parallel computation, numerical algorithms, and

applications to economics. The presentation will be on a high level of abstraction and theoretically rigorous. Although parallel processing plays a role in symbolic processing and artificial intelligence, such topics are beyond the scope of this presentation. We also do not address distributed processing, where the processors may be located a greater distance from one another, execute disparate tasks, and are characterized by communication that is not, typically, as reliable and as predictable as between parallel processors.

The chapter is organized as follows. In Section 2 we overview the technology of parallel computation in terms of hardware and programming languages.

In Section 3 we present some of the fundamental classes of problems encountered in economics and the associated numerical methodologies for their solution. In particular, we overview such basic problems as: nonlinear equations, optimization problems, and variational inequality and fixed point problems. In addition, we discuss dynamical systems. For each problem class we then discuss state-of-the-art computational techniques, focusing on parallel techniques and contrast them with serial techniques for illumination and instructive purposes. The techniques as presented are not machine-dependent, but the underlying parallelism, sometimes obvious, and sometimes not, is emphasized throughout.

In Section 4, we present applications of the classes of problems and associated numerical methods to econometrics, microeconomics, macroeconomics, and finance. Here we discuss the implementations of the parallel algorithms on different architectures and present numerical results.

2. Technology for Parallel Computation

In this section we further refine some of the ideas presented in the Introduction by focusing on the technology of parallel computation in terms of both hardware and software. In particular, we discuss some of the available parallel architectures as well as features of certain parallel programming languages. We then highlight some computer science issues of parallel algorithm development which reveal themselves during the presentation of the technology.

The field of parallel computing is quite new and the terminology has yet to be completely standardized. Nevertheless, the discussion that follows presents the fundamental terminology and concepts that are generally accepted.

2.1 Parallel Architectures

Although the speed at which serial computers do work has increased steadily over the last several decades, with a large part of the speedup process being due to the miniaturization of the hardware components, there are natural limits to speedup that are possible due to miniaturization. Specifically, two factors limit the speed at which data can move around a traditional central processing unit: the speed at which electricity moves along the conducting material and the length and thickness of the conducting material itself. Both of these factors represent physical limits and today's fastest computers are quickly approaching these physical constraints. Furthermore, as the miniaturization increases, it becomes more difficult to dissipate heat from the devices and more opportunities for electrical interference manifest themselves.

Other possibilities beyond the silicon technology used in miniaturization exist. These include optical computing and the development of a quantum transistor. Nevertheless, these technologies are not sufficiently advanced to be put into practical use.

Due to such limitations of serial computation, a paradigm for parallel computing emerged. Metaphorically speaking, if an individual or processing unit cannot complete the task in the required time, assigning, say, three individuals or units each a part of the task, will, ideally, result in the completion of the task in a third of the time.

The most prevalent approach to parallelism today uses the von Neumann or control-driven model of computation, which shall also be the focus here. Other approaches

presently under investigation are systolic, data flow, and neural nets. In addition, many of the terms that have been traditionally used for architectural classification of parallel machines can also be used to describe the structure of parallel algorithms and, hence, serve to suggest the best match of algorithm to machine.

Flynn's Taxonomy

One taxonomy of hardware is due to Flynn (1972) and although two decades old, it is still relevant in many aspects today. His classification of computer architectures is with respect to instruction stream and data stream, where intruction stream is defined as a sequence of operations performed by the computer and data stream is the sequence of items operated on by the instructions. In particular, the taxonomy may be depicted as follows:

$$\left\{\begin{matrix}\mathrm{S}\\ \overline{\mathrm{M}}\end{matrix}\right\} \quad \mathrm{I} \quad \left\{\begin{matrix}\mathrm{S}\\ \overline{\mathrm{M}}\end{matrix}\right\} \quad \mathrm{D},$$

where SI refers to single instruction, MI to multiple instruction, SD to single data, and MD to multiple data.

In single instruction, all processors are executing the same instruction at any given time where the instruction may be conditional. If there is more than a single processor, then this is usually achieved by having a central controller issue instructions. In multiple instruction, on the other hand, different processors may be simultaneously executing different instructions. In single data all processors are operating on the same data items at any given time, whereas in multiple data, different processors may be operating simultaneously on different data items.

Under this taxonomy, we have that a SISD machine is just the standard serial computer, but, perhaps, one with multiple processors for fault tolerance. A MISD machine, on the other hand, is very rare and considered to be impractical. Nevertheless, one might say that some fault-tolerant schemes that utilize different computers and programs to operate on the same input data are of this type.

A SIMD machine typically consists of N processors, a control unit, and an interconnection network. An example of the SIMD architecture is the Thinking Machines CM-2 Connection Machine, which will be discussed later in greater detail. A MIMD machine usually consists of N processors, N memory modules, and an interconnection network.

The multiple instruction stream model permits each of the N processors to store and execute its own program, in contrast to the single instruction stream model. An example of a MIMD architecture is the IBM SP1, which also will be discussed later more fully.

Shared Versus Distributed Memory

The second principal issue is that of memory. In a global or shared memory, there is a global memory space, accessible by all processors. Processors may, however, also possess some local memory. The processors in a shared memory system are connected to the shared (common global) memory by a bus or a switch. In local or distributed (message-passing) memory, all the memory is associated with processors. Hence, in order to retrieve information from another processor's memory, a message must be sent there. MIMD machines, for example, are organized with either the memory being distributed or shared.

Memory and bus contention must be considered in the case of algorithm development for a shared memory system, since caution must be taken when two processors try to simultaneously write to the same memory location. Distributed memory systems, on the other hand, avoid the memory contention problem, but since access to non-local data is provided by message passing between processors through the interconnection network, contention for message passing channels is, thus, of concern.

Interconnection Networks

Another common feature of parallel architectures is the interconnection network. Some basic interconnection networks are: a bus, a switching network, a point to point network, and a circuit-switched network. In a bus, all processors (and memory) are connected via a common bus or busses. The memory access in this case is fairly uniform but an architecture based on such an interconnection network is not very scalable due to contention. In a switching network, all processors (and memory) are connected to routing switches as in a telephone system. This approach is usually scalable. In a point to point network, the processors are directly connected to only certain processors and must go multiple hops to get to additional processors. In this type of network, one usually encounters distributed memory and this approach is also scalable.

Some examples of a point to point network are: a ring, a mesh, a hypercube, and

binary tree (also a fat tree). The processors' connectivity here is modeled as a graph in which nodes represent processors and edges the connections between processors. In a ring network, for example, the N processors are connected in a circular manner so that processor P_i is directly connected to processors P_{i-1} and P_{i+1}. In a mesh network, for example, the N processors of a two-dimensional square mesh are usually configured so that an interior processor $P_{i,j}$ is connected to its neighbors - processors $P_{i-1,j}$, $P_{i+1,j}$, $P_{i,j-1}$, and $P_{i,j+1}$. The four corner processors are each connected to their two remaining neighbors, while the other processors that are located on the edge of the mesh are each connected to three neighbors. A hypercube with N processors, on the other hand, where N must be an integral power of 2, has the processors indexed by the integers$\{0, 1, 2, \ldots, N-1\}$. Considering each integer in the index range as a $(\log_2 N)$-bit string, two processors are directly connected only if their indices differ by exactly one bit.

In a circuit-switched network, a circuit is sometimes established from the sender to the receiver with messages not having to travel a single hop at a time. This can result in significantly lower communication overhead, but at high levels of message traffic, the performance may be seriously degraded.

Desireable features of the interconnection network are the following: 1). any processor should be able to communicate with every other processor, 2). the networks should be capable of handling requests from all processors simultaneously with minimal delays due to contention, 3). the distance that data or messages travel should be of lower order than the number of processors, and 4). the number of wires and mesh points in the network should be of lower order than the square of the number of processors.

As discussed in Denning and Tichy (1990), many interconnection networks satisfy these properties, in particular, the hypercube. Although some computers utilize interconnection networks that do not satisfy these characteristics, they may, nevertheless, be cost-effective because the number of processors is small. Examples of computers that violate the fourth property above are the Cray X-MP and the Cray Y-MP, which utilize a crossbar switch as the interconnection network. These contain N^2 switch points and become unwieldy as the number of processors grows. The Sequent Symmetry and the Encore Multimax, for example, make use of a shared bus, which may result in violation of the second property above as the number of processors increases due to congestion on the

bus.

Granularity

Another term that appears quite frequently in discussions of parallel architectures is granularity. Granularity refers to the relative number and the complexity of the processors in the particular architecture. A fine-grained machine usually consists of a relatively large number of small and simple processors, while a coarse-grained machine usually consists of a few large and powerful processors. Speaking of early 1990s technology, fine-grained machines have on the order of 10,000 simple processors, whereas coarse-grained machines have on the order of 10 powerful processors (cf. Ralston and Reilly (1993)). Medium-grained machines have typically on the order of 100 processors and may be viewed as a compromise in performance and size between the fine-grained and coarse-grained machines.

Fine-grained machines are usually SIMD architectures, whereas coarse-grained machines are usually shared memory, MIMD architectures. Medium-grained machines are usually distributed memory, MIMD architectures. According to Ralston and Reilly (1993), by the mid 1990s, due to technological advances, one can expect that fine-grained machines will have on the order of a million of processors, coarse-grained machines will have on the order of one hundred processors, and medium-grained machines will have on the order of ten thousand processors.

Examples of Parallel Architectures

We now provide examples, selected from commercially available machines that illustrate some of the above concepts. The examples include fine-, coarse-, and medium-grained machines and SIMD and MIMD machines, along with a variety of interconnection networks. We first list some parallel computers and subsequently discuss several of them more fully. Examples of distributed memory, SIMD computers are: the Thinking Machines CM-1 and CM-2, the MasPar MP1, and the Goodyear MPP. Examples of shared memory, MIMD computers are: the BBN Butterfly, the Encore Multimax, the Sequent Balance/Symmetry, the Cray X/MP, Y/MP, and C-90, the IBM ES/9000, and the Kendall Square Research KSR1. Examples of distributed memory, MIMD computers are: the Cray T3D, the Intel iPSC series, the IBM SP1, the Intel Paragon, the NCUBE series, and the Thinking Machines CM-5.

It is also worth mentioning the INMOS transputer, with its name an amalgam of transistor and computer. It is a microprocessor, or family of microprocessors, which has been specially designed for the building of parallel machines. It consists of a RISC (Reduced Instruction Set Computer) processor, and its own high level programming language. It is well-suited to constructing MIMD architectures and can be used as either a single processor or as a network of processors. The RISC processor, since it maintains a minimum set of instructions, has room on its chip for other functions, and its design makes it easier to coordinate the parallel activity among separate units.

Those interested in additional background material on parallel architectures are referred to the books by Hockney and Jesshope (1981), DeCegama (1989) and Hennessy and Patterson (1990). For the historical role of parallel computing in supercomputing, see Kaufmann and Smarr (1993). For a comprehensive overview of parallel architectures and algorithms, see Leighton (1992). For an overview of parallel processing in general, see Ralston and Reilly (1993).

For illustrative purposes, we now discuss in greater detail several distinct parallel architectures that highlight the major issues discussed above. Some of these architectures are then used for the numerical computations presented in Section 4.

The CM-2

We begin with the fine-grained, massively parallel CM-2, manufactured by the Thinking Machines Corporation. The CM-2 is an example of a distributed memory, SIMD architecture with 2^{16}, that is, 65,536 processors in its full configuration, each with 8KB (kilobytes, that is, 2^{10} or 1,024 bytes) of local memory, and 2,048 Weitek floating point units. Other common configurations are CM-2's with 8K (8,192) or 32K (32,768) processors.

Each processor performs very simple, 1 bit, operations. Each processor can operate only on data that is in its own memory but the processors are interconnected so that data can be transferred between processors. Sixteen bit serial processors reside on a chip, and every disjoint pair of chips shares a floating point unit. The front-end system (often a SUN workstation or a VAX) controls the execution on a CM-2. Programs are developed, stored, compiled, and loaded on the front-end. The instructions issued by the front-end are sent to the sequencers which break down the instructions into low-level operations that

are broadcast to the processors. Program steps that do not involve execution on parallel variables are executed on the front-end. Communication between processors on a chip is accomplished through a local interconnection network. Communication between the 4,096 chips is by a 12-dimensional hypercube topology. The CM-2 has a peak performance of 32 GFLOPS (gigaflops or billions of floating point operations per second). See Thinking Machines Corporation (1990) for additional background on this architecture.

The CRAY X-MP/48

The CRAY series of supercomputers consists of coarse-grained shared memory machines where the vector processors may operate independently on different jobs or may be organized to operate together on a single job. A vector processor is a processor containing special hardware to permit a sequence of identical operations to be performed faster than a sequence of distinct operations on data arranged as a regular array.

The CRAY X-MP/48 system, manufactured by Cray Research, is a coarse-grained system with four vector processors and a total of 8 million 64-bit words. The peak performance of the system is .8 GFLOPS. Each processor contains 12 functional units that can operate concurrently. For additional information, see Cray Research, Inc. (1986).

The C90, also manufactured by Cray Research, contains 16 connected processing units, each of which is capable of performing a billion calculations a second. It is a shared memory, MIMD machine. Central memory is 2 GB (gigabytes, that is, 2^{30} or 1.024×10^9 bytes) and the solid state storage device serves as an extension of memory to provide an additional 4 GB. See Cray Research, Inc. (1993a) for further reading on this architecture.

The T3D exists as a 32-processor prototype, and can be expanded to 128 processors and, ultimately, 512 processors. It is a distributed memory, MIMD machine. Each of its processors is a DEC Alpha 64-bit microprocessor, with a theoretical peak of 150 MFLOPS. The topology of the T3D is that of a three-dimensional torus. Each processors can contain 16MB of memory. See Cray Research, Inc. (1993b) for additional information.

The KSR1

The Kendall Square Research KSR1 computer is a medium-grained, shared memory MIMD model whose shared memory is known as "ALLCACHE." The processors are interconnected in levels of rings, with proprietary processors, up to 32 in 1 level, 1066 in

2 levels, each with 512KB "subcache" and 32MB "local cache." It uses a Unix operating system, run in a distributed manner and includes provisions for dynamic load balancing and time-sharing nodes among different users. For additional information, see Kendall Square Research (1992).

The ES/9000

The IBM ES/9000 is a shared memory, MIMD machine, which includes a number of improvements in design and technology as compared to its predecessor, the IBM ES/3090. It consists of 6 processors, each a vector unit in its own right, and with 8 GB of storage. Each processor has a separate cache for instructions and data, thereby allowing for concurrent access of instructions and data. The ES/9000 also allows for high-speed data-handling.

For additional information on this architecture, see IBM Corporation (1992).

The SP1

The IBM SP1 is a distributed memory, MIMD architecture with "many" off-the-shelf IBM RS/6000 workstation processors in a single box. Hence, programs that have run on an IBM RS/6000 workstation can be easily ported to the SP1, since it has the same compilers available. The scalability of the SP1 lies in its switch technology. In the case of 64 processors, the grouping of the processors is in 4 racks of 16 processors each, with each rack containing its own switchboard, which handles all communication over the switch. A typical processor on the SP1 has 32KB of cache memory, 128MB of main memory, and 512MB of extended (virtual) memory.

For more information, see IBM Corporation (1993).

The Paragon

The Intel Paragon XP/S is a message passing, MIMD computer that can also support the SPMD (Single Program Multiple Data) programming style. It is sometimes referred to as a scalable heterogeneous multicomputer. It is compatible with the iPSC/860 family and has a 2D mesh interconnection network. Its "GP" nodes are involved in service and I/O and its "MP" nodes have four i860 XP processors operating in a shared memory implementation. We refer the interested reader to Intel Corporation (1992).

The CM-5

The Thinking Machines CM-5 is an example of a MIMD architecture. It is (typically) medium-grained with distributed memory. It can contain from 16 to 16K processors interconnected via a "fat tree" data network with a regular binary tree for the control network. It consists of processing nodes that are SPARC processors, each of which has 4 proprietary attached vector units. Each vector unit controls 8MB of memory. A group of nodes under the control of a single processor is called a partition and the control manager is called the partition manager. The nodes can be time-shared among different users (cf. Thinking Machines Corporation (1992a)). The programming model associated with this machine is referred to commonly as SPMD, Single Program Multiple Data, which can be viewed as an extension of the SIMD approach to a medium grain MIMD architecture.

2.2 Parallel Programming Languages and Compilers

There are two fundamental (and complementary) categories of parallel programming languages. The first category consists of explicitly parallel languages, that is, languages with parallel constructs, such as vector operations and parallel do-loops. Hence, in this category, the parallellism in a program must be specified explicitly by the programmer. Languages with explicitly parallel constructs can be further classified as being either low level or high level. The second category consists of languages in which the potential parallelism is implicit. For languages in this category, a parallelizing compiler must be available to determine which operations can be executed in parallel.

Most of the parallel languages developed to date have been Fortran extensions, due, in part, to the large investment in software development for numerical computations on serial architectures. For example, CM Fortran, developed for the Connection Machine, is an explicitly parallel programming language, and was influenced by Fortran 90. It is also referred to as a data level programming language and exhibits the themes common to such languages as: elementwise parallelism, replication, reduction, permutation, and conditionals (cf. Steele (1988)).

As an illustration, in elementwise parallelism, when one adds two arrays, one adds components elementwise. In terms of replication, one is interested in taking an amount of data and making more of it in, for example, a few to many case, which is an example

of "spreading" in Fortran. On the other hand, one has "reduction" when one takes many data items and reduces them to a few items. This occurs when one sums over many values, or takes the "max" or "min" of an array. One encounters a permutation when one does not change the amount of data but rearranges it in some fashion. CM Fortran (cf. Thinking Machines Corporation (1992b, 1993a)), for example, uses the Fortran 90 array features, whereas other data parallel languages usually incorporate a new data type. Once the datasets are defined in the form of arrays or structures, a single sequence of instructions causes the concurrent execution of the operations either on the full datasets or on selected portions.

We now present one of the above constructs in CM Fortran. Others are given in sample codes in Section 4. Consider the addition of two arrays `A` and `B`, each of dimension 200×200. The statement in CM Fortran is then given by: `C=A+B`. This statement is executed as a single statement and yields the 40,000 elements of the array `C` simultaneously. In the case where the number of array elements exceeds the number of processors, the compiler configures the algorithm for processing on "virtual processors" and assigns each element its own virtual processor.

In contrast, a serial Fortran 77 version yielding the values of `C` would consist of the following statements:

```
Do 10 i=1,200
Do 20 j=1,200
C(i,j)=A(i,j)+B(i,j)
20 Continue
10 Continue
```

The flow of control in a data parallel language is almost identical to that of its serial counterpart, without any code required to guarantee synchronization in the program as is needed in functional parallelism. In functional parallelism, for example, one may have to assign particular tasks to specific processors via specific parallel programming task allocation constructs and then wait for the tasks to be completed, also explicitly stated in the code, before reassignation. Such a feature is provided in Parallel Fortran (cf. IBM Corporation (1988)), which is used in the IBM 3090-400 and IBM ES/9000 MIMD architecture series (cf. IBM Corporation (1992)). In data level parallelism, in contrast, the compilers

and other system software maintain synchronization automatically. Furthermore, since the sequence of events is almost identical to those that would occur in a serial version of the program, program debugging, analysis, and evaluation is simplified.

There also exist Fortran extensions for programming shared memory, MIMD architectures. As an illustration, we provide the Fortran 77 code for the above matrix addition, embedded with Parallel Fortran constructs for task allocation, for the IBM 3090/600E, which can have up to six processors.

```
ntask=nprocs()
Do 5 i=1,ntask
originate any task itask(i)
5 Continue
Do 10 i=1,200
irow(i)=i
dispatch any task next(i), sharing(A1), calling add(irow(i))
10 Continue
wait for all tasks
```

This routine allocates the task of summing, term by term, the elements of the rows of A and B to an available processor. The summing is accomplished in the subroutine add, which shares the common A1 with the main routine, which, in turn, contains the elements of the three arrays A, B, and C. An IBM Parallel Fortran compiler would be needed for the compilation of the above code, illustrarting the complementary nature of explicit parallel programming and automatic parallelization.

The second category of languages, which relies on parallelizing compilers, is, indeed, common to the majority of shared memory MIMD machines and to supercomputers within this class. Languages in which potential parallelism is implicit are such common programming languages as the already-mentioned Fortran, Pascal, and C. The parallelizing compilers, for example, would automatically translate sequential Fortran 77 code into parallel form. Sequential Fortran code could, hence, in principle, be more easily ported across different parallel platforms with the availability of such compilers. These compilers have had their greatest success in translating Fortran do-loops into vector operations for execution on pipelined vector processors. Nevertheless, more research is needed in the area of

parallelizing compilers for distributed memory architectures.

Finally, it is also worth noting the Thinking Machine's CM-5, which incorporates a mix of parallel techniques. The extended model is referred to as coordinated parallelism. The CM-5 retains the positive features of a SIMD machine, in that it is good at synchronization and communication, and the positive feature of a MIMD machine, that of independent branching. In order to program the CM-5 in a MIMD style, one makes use of the CMMD library (cf. Thinking Machines Corporation (1993b)), which supports such operations as sending and receiving messages between nodes, and such global operations as scan, broadcast, and synchronization.

One must also be aware that there are libraries of software routines available for parallel architectures. For example, the CMSSL (Connection Machine Scientific Software Library) contains routines for solving systems of equations, ordinary differential equations, and linear programming problems (see, e.g., Thinking Machines Corporation (1992c)). We will illustrate the use of this library in an application in Section 4. In addition, there is now a utility known as CMAX, which enables the translation of serial Fortran 77 code to CM Fortran (see, e.g., Thinking Machines Corporation (1993c)).

The Intel Paragon and the Kendall Square Research KSR1 computers also support Fortran as well as C. The KSR1, in addition, supports Cobol, since many of its applications lie in database management. Of course, the Intel Paragon also makes use of message libraries. Hence, one sees that one no longer must learn different assembly languages in order to avail oneselves of parallel computation. Furthermore, it is expected that the major parallel architectures will also be supporting High Performance Fortran, thus making codes more portable across the different architectures (cf. High Peformance Fortran Forum (1992)).

2.3 Computer Science Issues in Parallel Algorithm Development

Before turning to the presentation of numerical methods for particular problem classes in Section 3, we briefly highlight issues revealed above which impact parallel algorithm development and which do not arise in serial computation. These issues should be kept in mind when reading the subsequent sections. The major issues are: decomposition, task scheduling, load balancing, and synchronization and communication. Finally, the

algorithm developer must have the target architecture in mind.

Decomposition

The first and foremost step in the development of any parallel program for the solution of a problem is to determine the level of decomposition that is possible. One typically first considers the decomposition of the problem itself from the highest level to the most refined. Naturally, one should exploit any obvious parallelism. In conjunction with problem decomposition, one should also consider domain decomposition, that is, whether or not one can break up the region of definition of the problem into smaller subregions. In many parallel numerical methods, as we shall see in Sections 3 and 4, problem and domain decomposition may go hand in hand. In addition, one must consider the decomposition of the data structures themselves, since, for example, data on a particular architecture may be stored in local memory, and how one designs the data structures will influence the architectural level solution of the problem.

Task Scheduling

After the problem is decomposed into subtasks, the subtasks must be allocated for completion by the available processors. This is easy to understand in the manager-worker parallel paradigm, in which the manager (a processor) partitions a task into subtasks and assigns them to workers (other processors). If the tasks are homogeneous, in that they take about the same amount of time to complete on the processors, and relatively independent, then the scheduling of the tasks can be accomplished by a deterministic or random assignment, or using some simple heuristic. If this is not the case, then it may be difficult to schedule the tasks efficiently. Hence, one should aim to decompose the problem into subproblems of relative difficulty and size. Such a mechanism will be illustrated in Section 4 in the context of a multicommodity trade problem.

Load Balancing

This issue brings us to load balancing, which can be achieved satisfactorily by breaking up the problem into similar subproblems. However, if the workload cannot be predicted well in advance, then one may have to make use of more advanced techniques than static load balancing provides. For example, one may attempt dynamic load balancing, adjusting

the workload update and task reassignment throughout the computation. This is, however, difficult to accomplish effectively and may require a great deal of experimentation.

Synchronization and Communication

After one has selected one (or more) decomposition strategies for a given problem, one needs to assess the amount of synchronization and communication that will be required. It terms of synchronization, one distinguishes between tightly and loosely synchronous and asynchronous. In a synchronous strategy one focuses on the elements of the data domain with the expectation that they are updated accordingly. With an asynchronous strategy there is no natural synchronization and one cannot determine what data will be available (and how old it may be) at a particular iteration increment. For aynchronous strategies it is often very difficult to establish convergence of the underlying algorithm. Communication requirements are distingushed between static, deterministic and dynamic, non-deterministic.

Target Machine

Finally, the importance of the target machine cannot be overestimated. In other words, the selection of an algorithm to solve a particular problem should be made with a view of the properties of the architecture on which the algorithm is to be implemented. For example, one should keep the following questions in mind. Is the intended architecture SIMD, MIMD, or a combination? Is the memory distributed or shared and what is the available size? What is the structure of the interconnection network? Does the parallel computer have vector capabilities? Does the architecture support any software libraries, which may contain frequently used routines that are optimized? Are there message passing utilities available? One must be cognizant of such issues in order to make the best mapping of an algorithm to a particular architecture for a specific problem.

Some Performance Measures

We conclude this section with a discussion of some performance measures. A common measure of the performance gain from a parallel processor is known as speedup. Roughly defined, it is the ratio of the time required to complete the job with one processor to the time required to complete the job with N processors. Perfect speedup, hence, would be N. The achievement of a perfect speedup, at least in principle, may be feasible in the following

situations: 1). in the case where each part of the problem is permanently assigned to a processor and each such subproblem is computationally equivalent, with the processors experiencing no significant delays in exchanging information and 2). in a machine where each subproblem can be dynamically assigned to available processors, it may be attained only as long as the number of subproblems ready for processing is at least N. The best that one can hope to achieve is speedup that is linear in the number of processors.

More rigorously speaking and a model that is often utilized in measurements and evaluations of parallel algorithms is the following. Let T_1^* be the time required to solve a particular problem using the best possible serial algorithm on a single processor and let T_N be defined as the amount of time required to solve the problem using a parallel algorithm implemented on N processors. Then the ratio

$$S_N = \frac{T_1^*}{T_N} \tag{2.1}$$

is known as the speedup of the algorithm, and the ratio

$$E_N = \frac{S_N}{N} = \frac{T_1^*}{NT_N} \tag{2.2}$$

as the efficiency of the algorithm. The above measures may also be evaluated as the functions of the size of the problem n, with $T_N = T_N(n)$. In the ideal situation, the speedup $S_N = N$ and the efficiency $E_N = 1$.

Since there may be difficulty in determining the best or optimal serial algorithm and, hence, T_1^*, this term is sometimes alternatively defined as: the time required by the best existing serial algorithm, the time required by a benchmark algorithm, or the time required for the problem using the particular parallel algorithm on a single processor of the parallel processing system. Note that the last definition yields a speedup measure which evaluates the parallelizability of the parallel algorithm but provides no information as to its overall efficiency vis a vis other existing algorithms. Reporting of numerical results on parallel architectures must, hence, clearly state precisely what measurement of T_1^* is being used for calculating speedup and efficiency.

Another measure, known as Amdahl's Law (see Amdahl (1967)), attempts to take into consideration the fact that parts of an algorithm (or code) may be naturally serial and not parallelizable, and, therefore, when a large number of processors may be available

the parallel parts of the program may be quickly completed, whereas the serial parts serve as bottlenecks. In particular, the law is expressed as follows

$$S_N \leq \frac{1}{f + \frac{(1-f)}{N}} \leq \frac{1}{f}, \quad \text{for all} \quad N, \tag{2.3}$$

where f denotes the fraction of the total computation that is inherently serial. As f approaches zero, the speedup approaches the idealized one.

Hence, based on Amdahl's Law, the maximum possible speedup, even with an unlimited number of processors, i.e., as $N \rightarrow \infty$, would be: $\frac{1}{f}$. Consequently, an application program that is 10% serial, as was found to be the case (at best) in many scientific programs in the 1960's, would run no more than ten times faster, even with an infinite number of processors. This law, however, fails to recognize that often a problem is scaled with the number of processors, and f as a fraction of size may be decreasing, that is, the serial part of the code may take a constant amount of the time, independent of the size of the problem. This argument is sometimes used as a refutation of Amdahl's Law.

The measure, mentioned already earlier, known as MFLOPS (or GFLOPS) is also considered a yardstick for algorithm (and architecture) evaluation. This measure may have to be determined by hand if there is not the software to compute it on a particular architecture.

All the above measures, although imperfect, can, nevertheless, provide useful guidelines.

3. Fundamental Problem Classes and Numerical Methods

In Section 2 the focus was on the technology and computer science aspects of parallel computing. In this section we turn to the mathematical programming and numerical analysis aspects of parallel computing. We first overview some of the fundamental mathematical problems encountered in economics and then discuss the numerical methods for their solution. In particular, we emphasize problem classes and associated computational schemes that can be (and have been) parallelized and that have been subjected to rigorous theoretical analysis. This presentation is by no means exhaustive but, instead, highlights problems that occur frequently in practice. The goal here is to present unifying concepts in an accessible fashion.

We begin with systems of equations, which have served as the foundation for many economic equilibrium problems. Moreover, computational schemes devised for this class of problems have also been generalized to handle problems with objective functions and inequalities. We then discuss optimization problems, both unconstrained and constrained, and consider the state-of-the-art of parallel algorithms for this problem class.

We subsequently turn to the variational inequality problem, which is a general problem formulation that encompasses a plethora of mathematical problems, including, among others, nonlinear equations, optimization problems, complementarity problems, and fixed point problems. A variety of serial and parallel decomposition algorithms are presented for this problem class.

The relationship between solutions to a particular dynamical system and the solutions to a variational inequality is recalled, as well as a general iterative scheme for the solution of such dynamical systems, which induces several well-known algorithms. For this problem class we also discuss parallel computing issues.

We first describe the classes of problems under consideration and then the associated numerical schemes.

3.1 Problem Classes

We briefly review certain problem classes, which appear frequently in economics, and then recall their relationship to the variational inequality problem.

For standardization of notation, let x denote a vector in R^n and F a given continuous

function from K to R^n, where K is a given closed convex set.

Systems of Equations

Systems of equations are common in economics, in particular, in the setting of defining an economic equilibrium state, reflecting that the demand is equal to the supply of various commodities at the equilibrium price levels, and in the formulation of macroeconometric models. Let $K = R^n$ and let $F : R^n \mapsto R^n$ be a given function. A vector $x^* \in R^n$ is said to solve a system of equations if

$$F(x^*) = 0. \tag{3.1}$$

This problem class is, nevertheless, not sufficiently general to guarantee, for example, that $x^* \geq 0$, which may be desireable in the case where the vector x refers to prices.

Optimization Problems

Optimization problems, on the other hand, consider explicitly an objective function to be minimized (or maximized), subject to constraints that may consist of both equalities and inequalities. Let f be a continuously differentiable function where $f : K \mapsto R$. Mathematically, the statement of an optimization problem is:

$$\text{Minimize} \quad f(x) \tag{3.2}$$

$$\text{subject to} \quad x \in K.$$

Note that in the case where $K = R^n$, then the above optimization problem is an unconstrained problem.

Optimization problems occur frequently in economics not only in microeconomics, such as in the theory of households or firms, but also in econometrics.

Complementarity Problems

Let R^n_+ denote the nonnegative orthant in R^n, and let $F : R^n \mapsto R^n$. Then the nonlinear complementarity problem over R^n_+ is a system of equations and inequalities stated as:

Find $x^* \geq 0$, such that

$$F(x^*) \geq 0 \quad \text{and} \quad F(x^*)^T \cdot (x^*) = 0. \tag{3.3}$$

Whenever the mapping F is affine, that is, whenever $F(x) = Mx + b$, where M is an $n \times n$ matrix and b and $n \times 1$ vector, the above problem is then known as the linear complementarity problem.

The Variational Inequality Problem

The finite-dimensional variational inequality problem, VI(F, K), is to determine a vector $x^* \in K$, such that

$$F(x^*)^T \cdot (x - x^*) \geq 0, \quad \text{for all} \quad x \in K, \tag{3.4}$$

where F is a given continuous function from K to R^n and K a given closed convex set.

Variational inequality problems have been used to formulate and solve a plethora of economic equilibrium problems ranging from oligopolistic market equilibrium problems to general economic equilibrium problems.

Dynamical Systems

Consider the dynamical system defined by the ordinary differential equation (ODE)

$$\dot{x} = \Pi(x, b(x)), \quad x(0) = x_0 \in K, \tag{3.5}$$

where given $x \in K$ and $v \in R^k$, the projection of the vector v at x is defined by

$$\Pi(x, v) = \lim_{\delta \to 0} \frac{(P(x + \delta v) - x)}{\delta},$$

and the orthogonal projection $P(x)$ with respect to the Euclidean norm by

$$P(x) = \arg\min_{z \in K} \|x - z\|. \tag{3.6}$$

The difficulty in studying the dynamical system (3.5), as discussed in Dupuis and Nagurney (1993), lies in that the right-hand side, which is defined by a projection, is discontinuous. Nevertheless, as established therein, the important qualitative properties of ordinary differential equations hold in this new, nonclassical setting. The projection ensures that the trajectory always lies within the feasible set K and, hence, satisfies the constraints. This would guarantee, for example, that if K was the nonnegative orthant, the production outputs in an oligopoly example would always be nonnegative; similarly, the prices in a Walrasian equilibrium problem would also always be nonnegative.

3.1.1 Relationship Between the Variational Inequality Problem and Other Problem Classes

We now review the fact that the variational inequality problem contains the above problem classes as special cases, discuss its relationship to the fixed point problem, and recall that its set of solutions corresponds to the set of solutions of the above dynamical system. For rigorous proofs, see Nagurney (1993).

For example, a system of equations (3.1) can be formulated as a variational inequality problem. Indeed, a vector $x^* \in R^n$ solves VI(F, R^n) if and only if $F(x^*) = 0$, where $F : R^n \mapsto R^n$.

Similarly, both unconstrained and constrained optimization problems can be formulated as variational inequality problems. Consider the optimization problem (3.2) with x^* as the solution. Then x^* is a solution of the variational inequality problem:

$$\nabla f(x^*)^T \cdot (x - x^*) \geq 0, \quad \text{for all} \quad x \in K.$$

On the other hand, if $f(x)$ is a convex function and x^* is a solution to VI$(\nabla f, K)$, then x^* is a solution to the above optimization problem.

If the feasible set $K = R^n$, then the unconstrained optimization problem is also a variational inequality problem.

The variational inequality problem, however, can be reformulated as an optimization problem, only under certain conditions. In particular, if we assume that $F(x)$ is continuously differentiable on K and that the Jacobian matrix

$$\nabla F(x) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix}$$

is symmetric and positive semi-definite, so that F is convex, then there is a real-valued function $f : K \mapsto R$ satisfying

$$\nabla f(x) = F(x)$$

with x^* the solution of VI(F, K) also being the solution of the optimization problem (3.2).

Hence, although the variational inequality problem encompasses the optimization problem, a variational inequality problem can be reformulated as a convex optimization

problem, only when the symmetry condition and the positive semi-definiteness condition hold. The variational inequality, therefore, is the more general problem in that it can also handle a function $F(x)$ with an asymmetric Jacobian.

The variational inequality problem contains the complementarity problem (3.3) as a special case. The relationship between the complementarity problem defined on the nonnegative orthant and the variational inequality problem is as follows. VI(F, R^n_+) and the complementarity problem defined above have precisely the same solutions, if any.

The relationship between the variational inequality problem and the dynamical system was established in Dupuis and Nagurney (1993). In particular, if one assumes that the feasible set K is a convex polyhedron, and lets $b = -F$, then the stationary points of (3.5), i.e., those that satisfy $\dot{x} = 0 = \Pi(x, -F(x))$, coincide with the solutions of VI(F, K).

This identification is meaningful because it introduces a natural underlying dynamics to problems which have, heretofore, been studied principally in a static setting at an equilibrium point.

Fixed Point Problems

We now turn to a discussion of fixed point problems in conjunction with variational inequality problems. In particular, we first recall that the variational inequality problem can be given a geometric interpretation. Let K be a closed convex set in R^n. Then for each $x \in R^n$, there is a unique point $y \in K$, such that

$$\|x - y\| \leq \|x - z\|, \quad \text{for all} \quad z \in K,$$

and y is the orthogonal projection of x on the set K, i.e., $y = P(x)$=arg $\min_{z \in K} \|x - z\|$.

Moreover, $y = P(x)$ if and only if

$$y^T \cdot (z - y) \geq x^T \cdot (z - y), \quad \text{for all} \quad z \in K$$

or

$$(y - x)^T \cdot (z - y) \geq 0, \quad \text{for all} \quad z \in K.$$

Recalling that for two vectors u, $v \in R^n$, the inner product $u \cdot v = |u| \cdot |v| \cdot \cos \theta$, and, hence, for $0 \leq \theta \leq 90^0$, $u \cdot v \geq 0$, then the last inequality above may be interpreted geometrically.

We now present a property of the projection operator that is useful both in qualitative analysis of equilibria and their computation. Let K again be a closed convex set. Then the projection operator P is nonexpansive, that is,

$$\|Px - Px'\| \leq \|x - x'\| \quad \text{for all} \quad x, x' \in R^n.$$

The relationship between a variational inequality and a fixed point problem can now be stated. Assume that K is closed and convex. $x^* \in K$ is a solution of the variational inequality problem if and only if x^* is a fixed point of the map

$$P(I - \gamma F) : K \mapsto K, \quad \text{for} \quad \gamma > 0 \quad \text{that is,} \quad x^* = P(x^* - \gamma F(x^*)).$$

3.1.2 Qualitative Properties of the Variational Inequality Problem

In this subsection, for completeness, we present certain qualitative results for the variational inequality problem. We also review certain properties and recall definitions which will be referred to in our discussions of the convergence of algorithms. The interested reader is referred to Kinderlehrer and Stampacchia (1980) for accompanying results in standard variational inequality theory.

Existence of a solution to a variational inequality problem follows from continuity of the function F entering the variational inequality, provided that the feasible set K is compact. Indeed, if K is a compact convex set and $F(x)$ is continuous on K, then the variational inequality problem admits at least one solution x^*.

In the case of an unbounded feasible set K, the existence of a solution to a variational inequality problem can, nevertheless, be established under the subsequent condition.

Let $\sum_R$ denote a closed ball with radius R centered at 0 and let $K_R = K \cap \Sigma_R$. K_R is then bounded.

By VI_R we denote the variational inequality problem

$$F(x_R^*)^T \cdot (y - x_R^*) \geq 0, \quad \text{for all} \quad y \in K_R.$$

In this case we have the following result. VI(F, K) admits a solution if and only if x_R^* satisfies $\|x_R^*\| < R$ for large enough R.

Although $\|x_R^*\| < R$ may be difficult to check, one may be able to identify an appropriate R based on the particular application.

Qualitative properties of existence and uniqueness become easily obtainable under certain monotonicity conditions. We first outline the definitions and then present the results.

Definition 3.1

$F(x)$ is monotone on K if

$$\left[F(x^1) - F(x^2)\right]^T \cdot (x^1 - x^2) \geq 0, \quad \text{for all} \quad x^1, x^2 \in K.$$

Definition 3.2

$F(x)$ is strictly monotone on K if

$$\left[F(x^1) - F(x^2)\right]^T \cdot (x^1 - x^2) > 0, \quad \text{for all} \quad x^1 \neq x^2, \quad x^1, x^2 \in K.$$

Definition 3.3

$F(x)$ is strongly monotone if

$$\left[F(x^1) - F(x^2)\right]^T \cdot (x^1 - x^2) \geq \alpha \|x^1 - x^2\|^2, \quad \text{for some} \quad \alpha > 0, \quad \text{and all} \quad x^1, x^2 \in K.$$

Definition 3.4

$F(x)$ is Lipschitz continous if there exists a positive constant L such that

$$\|F(x^1) - F(x^2)\| \leq L\|x^1 - x^2\|, \quad \text{for all} \quad x^1, x^2 \in K.$$

Recall now the following. Suppose that $F(x)$ is strictly monotone on K. Then the solution is unique, if one exists.

Monotonicity is closely related to positive definiteness and plays a role similar to that of convexity in optimization problems. Indeed, suppose that $F(x)$ is continuously differentiable on K and the Jacobian matrix

$$\nabla F(x) = \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_n} \end{pmatrix}$$

is positive semi-definite (positive definite), that is,

$$x^T \nabla F(x) v \geq 0, \quad \text{for all} \quad v \in R^n$$

$$(v^T \nabla F(x) v > 0, \quad \text{for all} \quad v \neq 0, v \in R^n),$$

then $F(x)$ is monotone (strictly monotone).

Under a slightly stronger condition, we have the following result. Assume that $F(x)$ is continuously differentiable on K and that $\nabla F(x)$ is strongly positive definite, that is,

$$v^T \nabla F(x) v \geq \alpha \|v\|^2, \quad \text{for all} \quad v \in R^n, \quad \text{for all} \quad x \in K.$$

Then $F(x)$ is strongly monotone.

The property of strong monotonicity guarantees both existence and uniqueness of a solution. In particular, if one assumes that $F(x)$ is strongly monotone, then there exists precisely one solution x^* to VI(F, K).

Assume now that $F(x)$ is both strongly monotone and Lipschitz continuous. Then the projection $P_K[x - \gamma F(x)]$ is a contraction with respect to x, that is, if we fix $\gamma \leq \frac{\alpha}{L^2}$ where α and L are the constants appearing, respectively, in the strong monotonicity and the Lipschitz continuity condition definitions. Then

$$\|P_K(x - \gamma F(x)) - P_K(y - \gamma F(y))\| \leq \beta \|x - y\|$$

for all $x, y \in K$, where

$$\beta(1 - \gamma\alpha)^{\frac{1}{2}} < 1.$$

It follows from this result and from the Banach fixed point theorem that the operator $P_K(x - \gamma F(x))$ has a unique fixed point x^*.

The above results are useful in establishing convergence of algorithmic schemes.

3.2 Algorithms

In this subsection we present some of the basic algorithmic schemes for the solution of the above problem classes. In particular, we focus on those algorithms, which have been successfully implemented in practice on both serial and parallel architectures, and that have been subject to theoretical analysis. Conditions for convergence are briefly discussed

with an aim towards accessibility. References where complete proofs can be obtained are included.

Many iterative methods for the solution of systems of equations, optimization problems, variational inequality and other problems, have the form

$$x^{\tau+1} = g(x^{\tau}), \quad \tau = 0, 1, \ldots, \tag{3.7}$$

where x^{τ} is an n-dimensional vector and g is some function from R^n into itself with components $\{g_1, g_2, \ldots, g_n\}$. For example, in the case where $g(x) = Ax + b$, where A is of dimension $n \times n$, and b is an n-dimensional vector, one obtains a linear iterative algorithm.

The principal iterations of the form (3.7) are the

Jacobi iteration:

$$x_i^{\tau+1} = g_i(x_1^{\tau}, \ldots, x_n^{\tau}), \quad i = 1, \ldots, n,$$

and the

Gauss-Seidel iteration:

$$x_i^{\tau+1} = g_i(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i^{\tau}, \ldots, x_n^{\tau}), \quad i = 1, \ldots, n.$$

As is well-known, the Gauss-Seidel algorithm incorporates the information as it becomes available, whereas the Jacobi method updates the iterates simultaneously. Hence, the Jacobi method is a natural parallel method. Indeed, each subproblem i, for the evaluation of $x_i^{\tau+1}$ can be allocated to a distinct processor for simultaneous solution. It is also worth noting that there are different Gauss-Seidel algorithms, depending on the specific order with which the variables are updated. Moreover, a Gauss-Seidel iteration may be totally unparallelizable as when each function g_i depends upon all of the components of the vector x, or it may be possible that, if this is not the case, component-wise updates may be done in parallel. Of course, one would want to then choose an ordering so that one could maximize the parallelism in any given iteration.

In our statements of the algorithms for the various classes of problems, for the sake of brevity, we present only the typical iteration. Of course, each algorithm must be suitably initialized and also convergence must be verified through an appropriate convergence criterion. This latter issue is discussed more fully in terms of specific applications in the numerical section.

3.2.1 Algorithms for Systems of Equations

The principal iterative techniques for solving systems of equations (cf. (3.1)) are the Gauss-Seidel and the Jacobi methods.

In the case where the system of equations itself is linear, say,

$$Ax = b,$$

under the assumption that A is invertible, one is guaranteed a unique solution x^*. If we write the i-th equation of $Ax = b$ as

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,$$

and assume that $a_{ii} \neq 0$, then the statement of the Jacobi algorithm for a typical iteration τ, and beginning with an initial vector $x^0 \in R^n$, would be

Jacobi iteration:

$$x_i^{\tau+1} = -\frac{1}{a_{ii}}\left[\sum_{j\neq i} a_{ij}x_j^{\tau} - b_i\right], \quad i = 1, \ldots, n,$$

and the statement of the Gauss-Seidel algorithm for a typical iteration τ:

Gauss-Seidel iteration:

$$x_i^{\tau+1} = -\frac{1}{a_{ii}}\left[\sum_{j<i} a_{ij}x_j^{\tau+1} + \sum_{j>i} a_{ij}x_j^{\tau} - b_i\right], \quad i = 1, \ldots, n.$$

Other variants of the above algorithms which make use of a relaxation parameter are the Jacobi Overrelaxation Method (JOR) and the Successive Overrelaxation Method (SOR). These algorithms, under an appropriate choice of relaxation parameter, denoted by γ (cf. Bertsekas and Tsitsiklis (1989)), often converge faster.

In particular, an iteration of the JOR algorithm (note the similarity to the Jacobi iteration) is given by

JOR iteration:

$$x_i^{\tau+1} = (1-\gamma)x_i^{\tau} - \frac{\gamma}{a_{ii}}\left[\sum_{j\neq i}^{n} a_{ij}x_j^{\tau} - b_i\right], \quad i = 1, \ldots, n,$$

whereas an iteration of the SOR algorithm (note the similarity to the Gauss-Seidel iteration) is given by

SOR iteration:

$$x_i^{\tau+1} = (1-\gamma)x_i^{\tau} - \frac{\gamma}{a_{ii}}\left[\sum_{j<i} a_{ij}x_j^{\tau+1} + \sum_{j>i} a_{ij}x_j^{\tau} - b_i\right], \quad i = 1, \ldots, n.$$

In the case where $\gamma = 1$, JOR and SOR collapse, respectively, to the Jacobi and Gauss-Seidel methods.

We now briefly discuss some convergence results. If the matrix A is row diagonally dominant, then the Jacobi method converges to the solution of the system of equations. The Gauss-Seidel method converges to the solution if the matrix A is symmetric and positive definite. Recall that a diagonally dominant matrix is also positive definite. Both the JOR algorithm and the SOR algorithm converge, under the same conditions as the Gauss-Seidel method, provided that the relaxation parameter γ is sufficiently small (and positive) in the case of JOR and in the range $(0, 2)$ for SOR.

All the above methods, nevertheless, share the desireable, especially from a practical point of view, property that, if they converge, then they converge to a solution.

3.2.2 Algorithms for Unconstrained Optimization

Here we consider algorithms for minimizing a continuous function $f : R^n \mapsto R$, in the absence of constraints (cf. (3.2)). In this case, $\nabla f(x^*) = 0$ for every vector x^* that minimizes f and, hence, the problem of minimizing f is related to solving the system $\nabla f(x) = 0$ of generally nonlinear equations, where $F(x) = \nabla f(x)$. In fact, the proofs of convergence of various schemes for solving linear equations, discussed above, also make use of this fact. Indeed, cf. Section 3.1.1, under certain assumptions, such as symmetry, in this case of A, one can reformulate the problem as an optimization problem, which would here take the form of a quadratic programming problem. This problem, in turn, would be strictly convex if A is assumed positive definite. Hence, in the proof one establishes that the objective function must decrease at each step. This is known as the "descent" approach to establishing convergence.

The statement of the nonlinear Jacobi algorithm for unconstrained optimization is given by the following expression.

Nonlinear Jacobi Method:

$$x_i^{r+1} = \arg\min_{x_i} f(x_1^r, \ldots, x_{i-1}^r, x_i, x_{i+1}^r, \ldots, x_n^r), \quad i = 1, \ldots, n.$$

The nonlinear Gauss-Seidel algorithm is defined by the following expression.

Nonlinear Gauss-Seidel Method:

$$x_i^{r+1} = \arg\min_{x_i} f(x_1^{r+1}, \ldots, x_{i-1}^{r+1}, x_i, x_{i+1}^r, \ldots, x_n^r), \quad i = 1, \ldots, n.$$

One assumes that a minimizing x_i^{r+1} always exists, and that the algorithms are initialized with an $x^0 \in R^n$.

The nonlinear Gauss-Seidel algorithm is guaranteed to converge to a solution x^* of (3.2) under the assumptions that f is continuously differentiable and convex, and f is a strictly convex function of x_i, when all the other components of the vector x are held fixed. Both algorithms are guaranteed to converge if the mapping T, defined by $T(x) = x - \gamma \nabla f(x)$, is a contraction with respect to a weighted maximum norm, where the weighted maximum norm $\|\cdot\|_\infty^w = \max_i |\frac{x_i}{w_i}|$. The sequence $\{x^r\}$ generated by either of these algorithms then converges to the unique solution x^* geometrically. The contraction condition would hold if the matrix $\nabla^2 f(x)$ satisfies a diagonal dominance condition (cf. Bertsekas and Tsitsiklis (1989)).

Note that different versions of the nonlinear algorithms are obtained if R^n can be decomposed into a Cartesian product: $\prod_{i=1}^s R^{n_i}$, where at each stage, the minimization is done with respect to the n_i-dimensional subvector x_i. Cartesian products will also play an important role in the construction of decomposition algorithms for both constrained optimization problems and variational inequality problems. These algorithms converge under analogous assumptions to those imposed previously.

Linearized counterparts of the above algorithms include a generalization of the JOR algorithm for linear equations, where

JOR Method:

$$x^{r+1} = x^r - \gamma [D(x^r)]^{-1} \nabla f(x^r)$$

where $\gamma > 0$, and $D(x)$ is a diagonal matrix whose i-th diagonal element is $\nabla^2 f(x)$, assumed nonzero.

A generalization of the SOR algorithm is given by

SOR Method:

$$x_i^{\tau+1} = x_i^{\tau} - \gamma \frac{\nabla_i f(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i^{\tau}, \ldots, x_n^{\tau})}{\nabla_{ii}^2 f(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i^{\tau}, \ldots, x_n^{\tau})}, \quad i = 1, \ldots, n.$$

Both the JOR algorithm, sometimes referred to as the Jacobi method, and the SOR algorithm, sometimes referred to as a Gauss-Seidel method, are guaranteed to converge to the unique solution x^*, provided that γ is chosen positive and small enough and ∇f is strongly monotone (cf. Definition 3.3). For this proof, rather than using the descent property, one uses a contraction approach (cf. Bertsekas and Tsitsiklis (1989)).

Both the nonlinear and linear Jacobi methods are easily parallelized with each subproblem i being allocated to a distinct processor i. Also, note that these are synchronous algorithms in that one obtains updates for all $\{x_i\}$, $i = 1, \ldots, n$, before proceeding to the next iteration.

If f is assumed to be twice continuously differentiable, then another important algorithm for the solution of nonlinear equations and optimization problems is Newton's method described below.

Newton's Method:

$$x^{\tau+1} = x^{\tau} - \gamma (\nabla^2 f(x^{\tau}))^{-1} \nabla f(x^{\tau}).$$

For additional results and theory with respect to algorithms for the solution of both nonlinear equations and unconstrained optimization problems, see Dennis and Schnabel (1983).

3.2.3 Algorithms for Constrained Optimization

Assuming now that the feasible set K is a Cartesian product, where $K = \prod_{i=1}^{z} K_i$, and each x_i is an n_i-dimensional vector, then one has natural extensions of the nonlinear algorithms introduced for the unconstrained case to optimization problems with constraints (cf. (3.2)). Indeed, the nonlinear Jacobi method is given by

Nonlinear Jacobi Method:

$$x_i^{\tau+1} = \arg\min_{x_i \in K_i} f(x_1^{\tau}, \ldots, x_{i-1}^{\tau}, x_i, x_{i+1}^{\tau}, \ldots, x_z^{\tau}), \quad i = 1, \ldots, z,$$

and the nonlinear Gauss-Seidel algorithm by

Nonlinear Gauss-Seidel Method:

$$x_i^{\tau+1} = \arg\min_{x_i \in K_i} f(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i, x_{i+1}^{\tau}, \ldots, x_z^{\tau}), \quad i = 1, \ldots, z.$$

Hence, the overall problem is decomposed into z smaller subproblems, each of which itself is a constrained optimization problem, but over a smaller and simpler feasible set.

Convergence of the iterates $\{x^\tau\}$ generated by the Gauss-Seidel algorithm to a minimizer of f over K is guaranteed under the assumptions that f is a continuously differentiable function, convex on K, and a strictly convex function of x_i when the other components of the vector x are held fixed. Under the very same conditions, hence, one was guaranteed convergence of the nonlinear Gauss-Seidel method for unconstrained optimization problems, where $K_i = R^{n_i}$.

Convergence of the nonlinear Jacobi method can be established under an appropriate contraction assumption on the mapping $x := x - \gamma \nabla f(x)$. The nonlinear Jacobi method can be implemented on a parallel architecture by allocating a distinct processor to each of the z subproblems for the computation of the respective x_i.

The linearized algorithms for unconstrained optimization are no longer valid for constrained optimization. This is due to the fact that, even if we begin within the feasible set K, an update can take us outside of the feasible set. A simple solution is to project back whenever such a situation occurs. Recall that the projection $P(x)$ was defined as: $P(x)$=arg $\min_{y \in K} \|x - y\|$.

In particular, we have the well-known gradient projection method, where an iterate is given by

Gradient Projection Method:

$$x^{\tau+1} = P(x^\tau - \gamma \nabla f(x^\tau))$$

with $\gamma > 0$, a positive stepsize.

Convergence conditions will now be briefly discussed. In particular, if $\nabla f(x)$ is strongly monotone and Lipschitz continuous (cf. Definition 3.4), and $f(x) \geq 0$, $\forall x \in K$, then if γ is selected to be small enough, the sequence $\{x^\tau\}$ defined by the above statement of the gradient projection algorithm converges to the solution x^* geometrically. What is important to note is that, although this linear algorithm is not at first appearance amenable

to parallelization, there may, nevertheless, be applications in which the realization of the above algorithm yields a natural decoupling. For example, this would be the case if the feasible set $K = \prod_{i=1}^{z} K_i$, with each $K_i = [0, \infty)$, where the projection would be obtained by projecting the i-th component of x on the interval $[0, \infty)$, which is simple and can be done independently and simultaneously for each component. A similar situation may arise in the case of the solution of dynamical systems, as we shall demonstrate in the numerical section.

Further, if K is a Cartesian product, one can consider a Gauss-Seidel version of the gradient projection algorithm defined by the iteration:

Gauss-Seidel Version of the Gradient Projection Method:

$$x_i^{\tau+1} = P[x_i^{\tau} - \gamma \nabla_i f(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i^{\tau}, \ldots, x_z^{\tau})], \quad i = 1, \ldots, z.$$

For supporting proofs of convergence of the above schemes, see Bertsekas and Tsitsiklis (1989).

The algorithms for optimization problems presented here have focused on problems where the constraint set is a Cartesian product. In the case where this does not hold one may, nevertheless, be able to exploit the underlying structure of a problem and take advantage of parallelism by transforming the problem in an appropriate fashion. One approach is to consider a dual optimization problem, which may be more suitable for parallelization than the original primal problem. A variety of decomposition approaches for large-scale problems are presented in Lasdon (1970) and parallel decomposition algorithms, in particular, are discussed in Bertsekas and Tsitsiklis (1989). Since many such algorithms are better understood in the context of a specific application, we defer a discussion along these lines until we consider a specific application in Section 4.

We also refer the interested reader to Lootsma and Ragsdell (1988) for an overview of parallel computation of nonlinear optimization problems and for a discussion of parallelization of the conjugate gradient method, variable-metric methods, and several decompositon algorithms. For an examination of dynamic programming and parallel computers, see Casti, Richardson, and Larson (1973) and Finkel and Mnaber (1987). Experimentation on the parallelization of combinatorial optimization algorithms can be found in Kindervater and Lenstra (1988).

3.2.4 Algorithms for Variational Inequality Problems

We now focus on the presentation of variational inequality algorithms, which may be applied for the computation of equilibria. In particular, we first present projection methods and then decomposition algorithms for when the variational inequality to be solved is defined over a Cartesian product of sets. We discuss decomposition algorithms of both the Jacobi and Gauss-Seidel type, the former being naturally implementable on parallel computer architectures. We don't present algorithms for complementarity problems, since these are special cases of variational inequality problems, and the theory of variational inequality algorithms is more developed. Moreover, we don't discuss fixed point algorithms since they may not be appropriate for large-scale problems.

Variational inequality algorithms resolve the variational inequality problem (3.4) into simpler variational inequality subproblems, which, typically, are optimization problems. The overall efficiency of a variational inequality algorithm, hence, will depend upon the optimization algorithm used at each iteration. The subproblems under consideration often have a special structure and special-purpose algorithms that exploit that underlying structure can be used to solve the embedded mathematical programming problems to realize further efficiencies.

3.2.4.1 Projection Methods

Projection methods resolve a variational inequality problem, typically, into a series of quadratic programming problems. They have been applied for the computation of a plethora of equilibrium problems (cf. Nagurney (1993)) and, although they were not developed as parallel computational procedures, per se, may, nevertheless, resolve the problem because of the underlying feasible set K, into (numerous) subproblems, which can then be solved simultaneously. The same characteristic was discussed in regards to the gradient projection method in subsection 3.2.3.

Projection Method:

$$x^{\tau+1} = P(x^{\tau} - \gamma G^{-1} F(x^{\tau}))$$

where G is a symmetric positive definite matrix, and $\gamma > 0$.

Convergence is guaranteed (cf. Bertsekas and Tsitsiklis (1989)) provided that the function F is strongly monotone (cf. Definition 3.3) and Lipschitz continuous (cf. Defini-

tion 3.4), for any $\gamma \in (0, \gamma^0]$, such that the mapping induced by the projection above is a contraction mapping with respect to the norm $\|\cdot\|_G$. The sequence $\{x^\tau\}$ generated by the projection algorithm then converges to the solution x^* of (3.4) geometrically.

In the case where the function F is no longer strongly monotone, but satisfies the less restrictive monotonicity condition (cf. Definition 3.1), and is also Lipschitz continuous, then the modified projection method of Korpelevich (1977) is guaranteed to converge to the solution of the variational inequality problem. If the function F is monotone, rather than strongly monotone, then a unique solution, however, is no longer guaranteed.

Modified Projection Method:

$$x^{\tau+1} = P(x^\tau - \gamma F(\bar{x}^\tau))$$

where $\bar{x}^\tau$ is given by

$$\bar{x}^\tau = P(x^\tau - \gamma F(x^\tau))$$

and γ, is, again, a positive scalar, such that $\gamma \in (0, \frac{1}{L}]$, where L is the Lipschitz constant in Definition 3.4. Note that here $G^{-1} = I$.

3.2.4.2 Decomposition Algorithms

Here we assume that the feasible set K is a Cartesian product, that is,

$$K = \prod_{i=1}^{z} K_i \tag{3.8}$$

where each $K_i \subset R^{n_i}$; $\sum_{i=1}^{z} n_i = n$; x_i denotes a vector in R^{n_i}, and $F_i(x) : K \mapsto R^{n_i}$ for each i.

Many economic equilibrium problems are defined over a Cartesian product set and, hence, are amenable to solution via variational inequality decomposition algorithms. For example, a variety of game theory problems would fall into this framework, where each player has his or her own objective function and feasible set, with the feasible set depending upon only the individual's particular strategies, and not on those of the other players. This would be the case in classical oligopolistic market equilibrium problems (cf. Cournot (1838), Gabay and Moulin (1980)). In addition, multicommodity spatial price equilibrium problems (cf. Takayama and Judge (1971), Dafermos (1986)) would also have a feasible

set defined as a Cartesian product, where each commodity would have to satisfy its own conservation of flow equations.

The appeal of decomposition algorithms lies in their particular suitability for the solution of large-scale problems. Moreover, parallel decomposition algorithms can be implemented on parallel computer architectures and further efficiencies realized.

We emphasize that for any given equilibrium problem there may be several alternative, albeit, equivalent, variational inequality formulations, which may, in turn, suggest distinct, novel, and not immediately apparent, decomposition procedures.

We present the nonlinear decomposition methods and then the linear decomposition methods. For each, we first present the Jacobi version and then the Gauss-Seidel version.

The statement of a typical iteration τ of the nonlinear Jacobi method is given by

Nonlinear Jacobi Method:

$$x_i^{\tau+1} = \text{solution of:} \quad F_i(x_1^\tau, \ldots, x_{i-1}^\tau, x_i, x_{i+1}^\tau, \ldots, x_z^\tau)^T \cdot (x_i' - x_i) \geq 0, \quad \forall x_i' \in K_i, \forall i.$$

A typical iteration of the nonlinear Gauss-Seidel method is given by

Nonlinear Gauss-Seidel Method:

$$x_i^{\tau+1} = \text{solution of:} \quad F_i(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i, x_{i+1}^{\tau-1}, \ldots, x_z^{\tau-1}) \cdot (x_i' - x_i) \geq 0, \quad \forall x_i' \in K_i, \forall i.$$

The linear Jacobi method, on the other hand, is given by the expression

Linear Jacobi Method:

$$x_i^{\tau+1} = \text{solution of:} \quad [F_i(x^\tau) + A_i(x^\tau) \cdot (x_i - x_i^\tau)]^T \cdot [x_i' - x_i] \geq 0, \quad \forall x_i' \in K_i, \forall i.$$

The linear Gauss-Seidel method is given by the expression

Linear Gauss-Seidel Method:

$$x_i^{\tau+1} = \text{solution of:}$$

$$[F_i(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i^\tau, \ldots, x_z^\tau) + A_i(x_1^{\tau+1}, \ldots, x_{i-1}^{\tau+1}, x_i^\tau, \ldots, x_z^\tau) \cdot (x_i - x_i^\tau)]^T \cdot [x_i' - x_i] \geq 0$$

$$\forall x_i \in K_i, \forall i.$$

There exist many possibilities for the choice of $A_i(\cdot)$. If $A_i(x^\tau) = \nabla_{x_i} F_i(x^\tau)$, then we have a Newton's method. If we let $A_i(x^\tau) = D_i(x^\tau)$, where $D_i(\cdot)$ denotes the diagonal part of $\nabla_{x_i} F_i(\cdot)$, then we have a linearization method. If $A_i(\cdot) = G_i$, where G_i is a fixed, symmetric and positive definite matrix, then we get a projection method.

Note that the variational inequality subproblems above should be easier to solve than the original variational inequality since they are smaller variational inequality problems, defined over smaller feasible sets. In particular, if in the linear methods we select the $A_i(\cdot)$ to be diagonal and positive definite, then each of the subproblems is equivalent to a separable quadratic programming problem with a unique solution (cf. Section 3.1.1).

The subproblems that must be solved at each iteration of the nonlinear methods are themselves variational inequality problems. Hence, an algorithm such as the projection method (cf. subsection 3.2.4.1) would have to be applied, where $\nabla f(x)$ would now be replaced by $F(x)$, and the relaxation parameter would need to lie in the range $(0, 1]$. See Dafermos (1983) and Nagurney (1993) for additional discussion of the projection method for variational inequality problems, as well as other algorithms, including the relaxation method.

The linear methods are appealing since each variational inequality subproblem may be expected to take on a simpler form for computational purposes than in the case of the nonlinear methods. This is especially true, as already mentioned above, if $A(\cdot)$ is selected to be diagonal.

We now present a convergence theorem for the above decomposition algorithms that is due to Bertsekas and Tsitsiklis (1989) (see, also, Nagurney (1993)).

Theorem 3.1

Suppose that the variational inequality problem (3.4) has a solution x^* and that there exist symmetric positive definite matrices G_i and some $\delta > 0$ such that $A_i(x) - \delta G_i$ is nonnegative definite for every i and $x \in K$, and that there exists a $\gamma \in [0, 1)$ such that

$$\|G_i^{-1}(F_i(x) - F_i(y) - A_i(y) \cdot (x_i - y_i))\|_i \leq \delta\gamma \max_j \|x_j - y_j\|_j, \quad \forall x, y \in K,$$

where $\|x_i\|_i = (x_i^T G_i x_i)^{\frac{1}{2}}$. Then the Jacobi and the Gauss-Seidel linear and nonlinear decomposition algorithms, with $A_i(x)$ being diagonal and positive definite, converge to the solution x^*.

Variational inequality theory was originally introduced by Hartman and Stampacchia (1966) for the study of partial differential equations, that arise principally in mechanics. Such problems, however, in contrast to the ones considered here, are infinite-dimensional. For the parallel solution of partial differential equations, see Ortega and Voigt (1985).

3.2.5 Algorithms for the Dynamical System

Although the dynamical system (3.5) provides a continuous adjustment process, a discrete time process is needed for actual computational purposes. Towards this end, in this subsection, we first review a general iterative scheme introduced in Dupuis and Nagurney (1993), which induces a variety of numerical procedures, all of which, in turn, are designed to estimate solutions to the variational inequality problem (3.4) and to trace the trajectory of the dynamical system from the initial state. We then present several schemes induced by the general iterative scheme. These schemes are not in themselves parallel. Nevertheless, since they are based on a projection operation, which in many applications takes on a very simple form that is decomposable in the variables, one oftentimes obtains a parallel scheme. Indeed, this will be illustrated in terms of concrete applications in Section 4.

The proposed algorithms for obtaining a solution to the variational inequality problem all take the form

$$x^{\tau+1} = P(x^{\tau} - a_{\tau}F_{\tau}(x^{\tau})), \tag{3.9}$$

where, without loss of generality, the "τ" denotes an iteration (or time period), $\{a_{\tau}, \tau \in T\}$ is a sequence of positive scalars, and the sequence of vector fields $\{F_{\tau}(\cdot), \tau \in T\}$ "approximates" $F(\cdot)$.

We now present the Euler-type method, which is the simplest algorithm induced by the above general iterative scheme.

Euler-Type Method:

In this case we have that

$$F_{\tau}(x) = F(x)$$

for all $\tau \in T$ and $x \in K$. This would correspond to the basic Euler scheme in the numerical approximation of standard ODEs.

Another method is

Heun-Type Method:

In this case we have that

$$F_\tau(x) = \frac{1}{2}\left[F(x) + F(x) + P(x - a_\tau F(x)))\right].$$

Finally, if the function F is defined in a sufficiently large neighborhood of K, another method is

Alternative Heun-Type Method:

In this case we set

$$F_\tau(x) = \frac{1}{2}\left[F(x) + F(x - a_\tau F(x))\right].$$

Other methods, which are induced by this general iterative scheme, include Runge-Kutta type algorithms.

We now consider a situation where the above schemes would be parallelizable. Suppose that the feasible set $K = \prod_{i=1}^{z} K_i$, where each $K_i = [0, \infty)$. Then it is easy to see that the expression (3.9) takes on the following simple closed form expression:

$$x_i^{\tau+1} = \max\{0, x^\tau - a_\tau F_\tau(x^\tau)\}, \quad i = 1, \ldots, z.$$

All of the $x_i^{\tau+1}$'s, hence, can be updated in parallel.

We now give the precise conditions for the general convergence theorem and present its statement. For proofs, see Dupuis and Nagurney (1993).

Assumption 3.1

Fix an initial condition $x^0 \in K$ and define the sequence $\{x^\tau, \tau \in T\}$ by (3.9). Assume the following conditions.

1. $\sum_{i=0}^{\infty} a_i = \infty, \quad a_i > 0, \quad a_i \to 0$ as $i \to \infty$.

2. $d(F_\tau(x), \bar{F}(x)) \to 0$ uniformly on compact subsets of K as $\tau \to \infty$, where $d(x, A) = \inf\{\|x - y\|, y \in A\}$, and the overline indicates closure.

3. Define ϕ_y to be the unique solution to $\dot{x} = \Pi(x, -F(x))$ that satisfies $\phi_y(0) = y \in K$. The $\omega-$limit set

$$\cup_{y \in K} \cap_{t \geq 0} \overline{\cup_{s \geq t} \{\phi_y(s)\}}$$

is contained in the set of stationary points of $\dot{x} = \Pi(x, -F(x))$.

4. The sequence $\{x^\tau, \tau \in T\}$ is bounded.

5. The solutions to $\dot{x} = \Pi(x, -F(x))$ are stable in the sense that given any compact set K_1 there exists a compact set K_2 such that $\cup_{y \in K \cap K_1} \cup_{t \geq 0} \{\phi_y(t)\} \subset K_2$.

The assumptions are phrased as they are because they describe more or less what is needed for convergence, and because there are a number of rather different sets of conditions that imply the assumptions.

Theorem 3.2

Let S denote the solutions to the variational inequality (3.4), and assume Assumption 3.1 and Assumption 3.2, where

Assumption 3.2

There exists a $B < \infty$ such that the vector field $-F : R^n \mapsto R^n$ satisfies the linear growth condition: $\| - F(x)\| \leq B(1 + \|x\|)$ for $x \in K$, and also

$$\langle -F(x) + F(y), x - y \rangle \leq B\|x - y\|^2$$

for all $x, y \in K$.

Suppose $\{x^\tau, \tau \in T\}$ is the scheme generated by (3.9). Then $d(x^\tau, S) \to 0$ as $\tau \to \infty$.

Corollary 3.1

Assume the conditions of Theorem 3.2, and also that S consists of a finite set of points. Then $\lim_{\tau \to \infty} x^\tau$ exists and equals a solution to the variational inequality (3.4).

The above classes of problems and accompanying numerical methods were selected for their general applicability with an eye towards unifying principles. In the subsequent section we focus on applications and numerical results that help to illustrate and synthesize the computer science and mathematical programming principles of parallel computing discussed thus far.

4. Applications and Numerical Results

In this section we discuss both applications and numerical results. We begin with an application of systems of equations, and also discuss applications of optimization problems, variational inequality problems, and dynamical systems. The applications are drawn from econometrics, macroeconomics, and finance.

4.1 Nonlinear Equations

In this subsection we discuss an application of nonlinear equations (cf. (3.1)) that illustrates the parallel computation of the solution via the Jacobi and the Gauss-Seidel methods using a software library.

4.1.1 Econometric Model Simulation

Nonlinear equations are used in the formulation of macroeconometric systems. Such problems can be very large, especially when one wishes to solve the same model repeatedly for different data sets, as would be the case, for example, in stochastic simulation and optimal control. Indeed, such problems were some of the first economic problems that were solved using supercomputers with vectorization (see, e.g., Amman (1985), Ando, Beaumont, and Ando (1987), Petersen (1987), Petersen and Cividini (1989), and Amman (1989)).

Recently, Gilli and Pauletto (1993) considered the solution of a system of equations (3.1) consisting of linear and nonlinear equations on the CM-2 architecture. The model that they solved was a macroeconometric model of the Japanese economy, developed at the University of Tsukuba and consisting of 98 equations and 53 exogenous variables. The model was solved for ten time periods from 1973 to 1982.

The model, when put into block recursive form, exhibited a pattern common to macroeconometric models, in that a large fraction of the variables (77 of them) lay in one interdependent block, and were both preceded and followed by recursive equations. 6 variables were defined recursively before the block, followed by 15 variables that did not feed back on the block.

The authors studied the ordering of the equations for the purposes of the Gauss-Seidel algorithm through the use of a DAG (directed acyclic graph) in order to try and achieve the highest possible speedup. Both the Gauss-Seidel and the Jacobi method converged for

the model, the latter requiring 4.5 seconds, and the former .18 seconds using 8K (8,192) processors of the CM-2.

The authors then proceeded to repeatedly solve the same model, but using 8,192 different datasets, corresponding to 8,192 different sets of exogenous variables. The number of datasets was selected to correspond to the number of processors in order to yield the best speedup possible. The Gauss-Seidel algorithm required 22.2 seconds on the CM-2 using the convergence tolerance $|\frac{x_i^{\tau+1}-x_i^{\tau}}{x_i^{\tau}}| \leq \epsilon$, for all i, with $\epsilon = .001$. It required 1,109 seconds on a Sun ELC, yielding a speedup of 50. Since the convergence verification step itself was found to be time-consuming, a modification of it reduced the CPU time on the CM-2 to 12.7 seconds and on the Sun to 863 seconds, yielding an improved speedup of 68. The authors made use of the library, CMSSL (cf. Thinking Machines Corporation (1992c)), which contains routines for solving systems of equations.

4.2 Optimization Problems

In this subsection we focus on constrained optimization problems (cf. (3.2)), in particular, portfolio optimization problems and, subsequently, an estimation problem known as the constrained matrix problem. Both of these constrained optimization problems are quadratic programming problems. For the portfolio optimization problem we utilize the gradient projection method in which we embed a special-purpose algorithm for the solution of the simpler subproblems. For the constrained matrix problem, we apply a dual method which has been specifically developed for this problem and exploits its special structure. Since the resulting subproblems are of the same structure as those encountered in the application of the gradient projection method to the portfolio optimization problem, the same special purpose algorithm is applied at each iteration.

4.2.1 Portfolio Optimization Problems

As mentioned in the Introduction, portfolio optimization problems stimulated the interest in the development of the area of mathematical programming known as quadratic programming.

Recall the classical portfolio optimization problem (cf. Markowitz (1959) and Sharpe (1970)) where there are n financial instruments, x denotes the n-dimensional vector of shares of the instruments, Q is the variance-covariance matrix of dimension $n \times n$, and r

is the n-dimensional vector of expected returns on the individual instruments. Then the portfolio optimization problem may be formulated as follows:

$$\text{Minimize} \quad f(x) = x^T Q x - r^T x \tag{4.1}$$

subject to

$$\sum_{j=1}^{n} x_j = 1 \tag{4.2}$$

$$x_j \geq 0, \quad \text{for all} \quad j = 1, \ldots, n, \tag{4.3}$$

where the objective function denotes the risk minus the expected returns.

One may also introduce a risk parameter λ in which case the objective function (4.1) is modified to

$$\text{Minimize} \quad f(x) = x^T Q x - \lambda r^T x \tag{4.4}$$

and the full problem also incorporates the above constraints. The variational inequality formulation of this problem is given by

$$\left[2Qx^* - \lambda r^T\right]^T \cdot [x - x^*] \geq 0, \quad \forall x \in K, \tag{4.5}$$

where the feasible set K consists of the constraint (4.2) and the nonnegativity constraints (4.3).

An application of the gradient projection method discussed in subsection 3.2.3 resolves this problem into simpler quadratic programming problems, where the quadratic matrix at each iteration is now the diagonal identity matrix and where at iteration τ the subproblem is given by:

$$\text{Minimize}_{x^\tau \in K} \quad {x^\tau}^T x^\tau + (\gamma(2Qx^\tau - \lambda r^T) - x^{\tau-1})^T x^\tau. \tag{4.6}$$

In applications one may wish to vary the λ over a horizon in which case we have the problem

$$\text{Minimize} \quad f(x) = \sum_{i=1}^{m} {x^i}^T Q x^i - \lambda^i r^T x^i \tag{4.7}$$

subject to

$$\sum_{j=1}^{n} x_j^i = 1, \quad i = 1, \ldots, m, \tag{4.8}$$

$$x_j^i \geq 0, \quad \text{for all} \quad i = 1, \ldots, m; \quad j = 1, \ldots, n, \tag{4.9}$$

where x^i here denotes the n-dimensional vector with components $\{x_1^i, \ldots, x_n^i\}$ corresponding to the shares associated with the problem λ^i.

Observe that this problem may be decomposed into m subproblems, each of the form given by (4.4) with a distinct λ. In fact, an even finer decomposition (on the level of $m \times n$ is possible with an appropriate implementation of the exact equilibration algorithm, which is discussed (in a more general context) in subsection 4.2.2.

The dataset that we utilized consisted of a variance-covariance matrix that was estimated using the Standard & Poor's index consisting of 500 firms. The data consisted of monthly data from 1986 through 1992 and the resulting estimated Q matrix was of dimension 500×500.

The system utilized for the implementation of the gradient projection method was the CM-2 with a SUN as the front-end. The parameter γ was set to .001 and the convergence tolerance ϵ was set equal to .0001. The convergence criterion was: $|x^\tau - x^{\tau-1}| \leq \epsilon$. The gradient projection method with the embedded exact equilibration scheme was implemented in CM Fortran.

First, the single portfolio optimization problem with $\lambda = 1$ was solved; this problem was equivalent to (4.1) subject to (4.2) and (4.3). Subsequently, λ was varied from 1 to 100 (cf. (4.7)) in increments of 2 yielding 51 portfolio optimization problems with a total of 25,500 variables.

The single problem with $\lambda = 1$ required 252 iterations for termination, whereas the 51 problems, with $\lambda^1 = 1, \lambda^2 = 3, \ldots, \lambda^{51} = 100$, required a total of 312 iterations. The CPU times on the CM-2 required for convergence are reported in Table 4.1.

Table 4.1
Portfolio Optimization
CPU Times in Seconds

Example	8K	16K	32K
single problem	365.28	238.16	—
51 problems	962.92	694.85	421.40

We did not solve the single problem using 32K processors since it had only 500 variables. In regards to the relative times on 8K processors, the 51 problems required less than 3 times the amount of CPU time as did the single problem. Indeed, when 32K processors were used the 51 problems required only approximately 20% more CPU time than the single problem required on 8K processors.

It is worth mentioning that multi-sector, multi-instrument general financial equilibrium problems, formulated as variational inequality problems (cf. Nagurney, Dong, and Hughes (1992), Nagurney (1994)) can be decomposed into subproblems of the form considered here and, hence, at least in principle, are also amenable to massively parallel computation.

4.2.2 Constrained Matrix Problems

Constrained matrix problems arise in numerous applications, such as the estimation of input/output tables, social accounting matrices, migration tables, and origin/destination tables in transportation (see, e.g., Bacharach (1970)). These problems can also be very large-scale in practice and are usually formulated as optimization problems. Here we shall consider a special-purpose algorithm for the solution of the problem which allows for parallel computation. The algorithm is a dual method that decomposes the problem into many simpler subproblems, each of which can then be solved explicitly and in closed form.

In this subsection we briefly review the constrained matrix problem with known row and column totals under consideration here. For the formulation of the general quadratic constrained matrix problem with unknown row and column totals and other variants, we refer the reader to Nagurney and Eydeland (1992), and the references therein.

In particular, we consider the diagonal constrained matrix problem, which is formulated as a minimization of the weighted squared sums of the deviations. We denote the given $m \times n$ matrix by $X^0 = (x_{ij}^0)$, and the matrix estimate by $X = (x_{ij})$. Let s_i^0 denote the known row i total, and s_i the estimate of the row i total. Let d_j^0 denote the known column j total, and d_j the estimate of the column j total. We assume that the γ_{ij} elements are all positive.

The diagonal quadratic constrained matrix problem is given by:

$$\text{Minimize} \quad f(x) = \sum_{i=1}^{m} \sum_{j=1}^{n} \gamma_{ij} (x_{ij} - x_{ij}^0)^2 \tag{4.10}$$

subject to the row constraints

$$\sum_{j=1}^{n} x_{ij} = s_i^0, \quad i = 1, \ldots, m \tag{4.11}$$

and the column constraints

$$\sum_{i=1}^{m} x_{ij} = d_j^0, \quad j = 1, \ldots, n, \tag{4.12}$$

where

$$x_{ij} \geq 0, \quad \text{for all} \quad i, j. \tag{4.13}$$

Note that this problem is of the form (3.2), where the feasible set K is defined by the set of x that satisfy constraints (4.11)-(4.13).

For generalizations of this model to the estimation of financial flow of funds accounts, see Hughes and Nagurney (1992) and Nagurney and Hughes (1992).

4.2.2.1 The Splitting Equilibration Algorithm and the Exact Equilibration Algorithm

Neither a Gauss-Seidel algorithm nor a Jacobi algorithm can be applied for the solution of this problem because the feasible set K here is not a Cartesian product, that is, of the form (3.8). Nevertheless, the problem has alot of structure that can be exploited for parallel computation. In particular, one can see that if the objective function was subject to either only the constraints (4.11) and (4.13), or (4.12) and (4.13), then each of the m, respectively, n subproblems could be solved simultaneously. Note, for example, the similarity of this optimization problem to the portfolio optimization problem (4.7) subject to constraints (4.8) and (4.9), in which case s_i^0 would be equal to 1 but although Q is no longer diagonal in the portfolio optimization problem, the gradient projection method approximates the problem by making use of the diagonal identity matrix at each iteration. Hence, the decomposed subproblems have essentially the same structure and can be solved by the same suitable algorithm.

The algorithm, known as the Splitting Equilibration Algorithm (SEA) (cf. Nagurney and Eydeland (1992)), computes a solution to the quadratic programming problem (4.10), subject to constraints (4.11) through (4.13), by first considering a modification of the

objective function subject to only the row constraints (4.11), and then by considering a modification of the objective function subject to only the column constraints (4.12). The former problem is referred to as the Row Equilibration Step, whereas the latter problem to as the Column Equilibration Step. The algorithm can be interpreted and analyzed as a dual method.

The simplicity of the procedure lies in that each of the row/column subproblems, because of their special structure, can be solved exactly, and in closed form, using exact equilibration. Exact equilibration algorithms were originally introduced by Dafermos and Sparrow (1969), and then later generalized and theoretically analyzed in Eydeland and Nagurney (1989). The massively parallel implementation of the Splitting Equilibration Algorithm (as the implementation of the gradient projection method for the portfolio optimization problem) depends crucially on the massively parallel implementation of the (row/column) exact equilibration algorithm.

The statement of SEA is as follows:

The Splitting Equilibration Algorithm

Step 0: Initialization Step:

Let $\mu^1 \in R^n = 0$. Set $\tau := 1$.

Step 1: Row Equilibration:

Find $X(\mu^\tau)$ such that

$$X(\mu^\tau) \rightarrow \mathrm{Min}_x f(x) - \sum_{j=1}^{n} \mu_j^\tau (\sum_{i=1}^{m} x_{ij} - d_j^0) \tag{4.14}$$

subject to

$$\sum_{j=1}^{n} x_{ij} = s_i^0, \quad i = 1, \ldots, m, \tag{4.15}$$

$$x_{ij} \geq 0, \quad \text{for all} \quad i, j.$$

Compute the Lagrange multipliers according to:

$$\lambda_i^{\tau+1} = 2\gamma_{ij} X_{ij}(\mu^\tau) - 2\gamma_{ij} x_{ij}^0 - \mu_j^\tau, \quad \text{for} \quad i = 1, \ldots, m.$$

Step 2: Column Equilibration

Find $X(\lambda^{\tau+1})$ such that

$$X(\lambda^{\tau+1}) \rightarrow \mathrm{Min}_x f(x) - \sum_{i=1}^{m} \lambda_i^{\tau+1} (\sum_{j=1}^{n} x_{ij} - s_i^0) \tag{4.16}$$

subject to

$$\sum_{i=1}^{m} x_{ij} = d_j^0, \quad j = 1, \ldots, n, \tag{4.17}$$

$$x_{ij} \geq 0, \quad \text{for all} \quad i, j.$$

Compute the Lagrange multipliers according to:

$$\mu_j^{\tau+1} = 2\gamma_{ij} X_{ij}(\lambda^{\tau+1}) - 2\gamma_{ij} x_{ij}^0 - \lambda_i^{\tau+1}, \quad \text{for} \quad j = 1, \ldots, n.$$

Step 3: Convergence Verification

If $|(\sum_j x_{ij}(\lambda^{\tau+1}) - s_i^0)/s_i^0| \leq \epsilon$, for all i, terminate; else, set $\tau := \tau + 1$, and go to Step 1.

We now present an algorithm for the solution of each of the row/column subproblems with special structure. The notable feature of this procedure is that it lends itself to a massively parallel implementation. For simplicity, we develop the presentation of the exact equilibration procedure in the context of the column equilibration step.

In particular, in the column equilibration step at iteration τ, we are interested in computing for each subproblem j, the flows $x_{1j}, \ldots, x_{mj}$ from the rows $1, \ldots, m$ to the column j that satisfy the constraint (4.17) and the following optimality/equilibrium conditions:

$$g_{ij} x_{ij} + h_{ij} \begin{cases} = \mu_j, & \text{if} \quad x_{ij} > 0 \\ \geq \mu_j, & \text{if} \quad x_{ij} = 0 \end{cases} \tag{4.18}$$

where the $g_{ij} = 2\gamma_{ij}$ terms, for $i = 1, \ldots, m$, are all greater than zero and $h_{ij} = -2\gamma_{ij} x_{ij}^0 - \lambda_i^{\tau+1}$, for $i = 1, \ldots, m$. The term μ_j is simply the Lagrange multiplier corresponding to the constraint (4.17) (where the superscript $\tau + 1$ has been removed), and not known a priori. The algorithm below computes the solution to the above system (4.18) in closed form. It can then be applied to solve each of the n column subproblems.

Exact Equilibration

(i). Sort the h_{ij}'s; $i = 1, \ldots, m$, in nondescending order and relabel the h_{ij}'s accordingly. Define $h_{m+1,j} = \infty$. Set $v := 1$.

(ii). Compute

$$\mu_j^v = \frac{\sum_{i=1}^{v} h_{ij}/g_{ij} + d_j^0}{\sum_{i=1}^{v} 1/g_{ij}}. \tag{4.19}$$

If $h_{vj} < \mu_j^v \leq h_{v+1,j}$, then stop; $s' := v$, and go to (iii). Otherwise, set $v := v + 1$, and go to (ii).

(iii). Set

$$x_{ij} = \frac{\mu_j^{s'} - h_{ij}}{g_{ij}}, \quad i = 1, \ldots, s'$$

$$x_{ij} = 0, \quad i = s' + 1, \ldots, m.$$

Note that $\mu_j^{\tau+1}$ is then equal to the $\mu_j^{s'}$ computed above.

In an analogous manner one can construct an exact equilibration algorithm for solving the i-th row equilibration subproblem (4.14), subject to constraint (4.15) for the particular row. Note that the algorithm should then be applied for the solution of all the m row subproblems.

As described above SEA decomposes the constrained matrix problem into m row subproblems, each of which can be solved independently and simultaneously on a distinct processor using exact equilibration, and into n column subproblems, each of which can also be solved independently and simultaneously. In this context, hence, if $m = n$ then at most m processors would be used for the parallel implementation; this is, indeed, the case with a coarse-grain architecture. However, in the next subsection we will show how a massively parallel implementation of SEA with the exact equilibration algorithm exploits all $n \times n$ processors, if such an architecture is available.

4.2.2.2 The Massively Parallel Implementation of SEA

The massively parallel implementation of SEA was earlier reported in Kim and Nagurney (1993). The language that was used for the implementation was CM Fortran and the architecture, the CM-2.

We now briefly describe some of the instrinsic functions of CM Fortran that make it very well-suited for implementing the exact equilibration algorithm. For example, the intrinsic function `cmf_order` sorts elements of a matrix either row - wise or column - wise and returns the indices. The `minval` and `maxval` functions, in turn, return the smallest, respectively, largest element of a row or a column in an array. The `transpose` feature of a matrix, in turn, is useful in minimizing the cost of communication between processors in which the data elements are located.

Since matrix operations in CM Fortran must be conformable, i.e., the operated on matrices must be of the same dimensions, one may need to change a matrix into a vector, or vice versa; for such transformations the functions **pack** and **unpack** are very useful. Also one may use the **spread** command to replicate a vector into a matrix.

Finally, we note the availability of logic statements, such as the **where, else, end** statement that checks conditions on vector/matrix elements in parallel.

Here we consider the estimation of an input/output table. Before presenting the numerical example, we focus on the critical implementation issues.

Recall that SEA decomposes the constrained matrix problem into row subproblems and column subproblems. Hence, in an $n \times n$ problem there would be n row subproblems to be solved and then n column subproblems, until convergence. In particular, the solution of each of the n subproblems of the form (4.18), which consisted of n unknown x_{ij} variables, was carried out by using n of the processors to first compute the μ_j^v given in equation (4.19), for $v = 1, \ldots, n$. A **shift** command was then utilized in order to bring the neighboring $h_{vj}, h_{v+1,j}$ values to the same location, in order to minimize the communication. The $h_{vj} < \mu_j^v \leq h_{v+1,j}$ check condition was implemented using the **where, else, end** construct. All n column problems were solved in the same fashion, simultaneously. The x_{ij}'s for $i = 1, \ldots, n$; $j = 1, \ldots, n$ were then updated, also simultaneously.

We report now the results of both the implementation of SEA on the CM-2 and on the IBM 3090/600.

For the parallel version of SEA on the IBM 3090/600E we utilized as the base the serial FORTRAN code developed on this machine and added the Parallel Fortran (PF) constructs in order to handle the task allocation, that is, the assignment of each of the n row / n column subproblems to the six CPU's. The conversion of the serial Fortran

code to this parallel code was relatively straightforward in that only task origination statements, dispatch statements that allocated a row/column subproblem to the next available processor, a waiting statement for synchronization, and task termination statements had to be added to the original serial code.

We now present the results of our implementations on the two architectures. The convergence criterion was set at $\epsilon = .01$. The weights, the γ_{ij}'s were set to $\frac{1}{x^0_{ij}}$ for $x^0_{ij} > 0$, and to 1, otherwise.

In Table 4.2 we present the results of the computations on the CM-2 system for a dataset based on an input/output matrix, IO72b, consisting of 485 rows and 485 columns and representing a dataset of a 1972 input/output matrix for the US. This problem consisted of 235,225 variables. The problem was solved using 8K processors, 16K processors, and, finally, 32K processors.

Table 4.2

Constrained Matrix Problem

Example IO72b (485 rows × 485 columns)

# of Physical Processors	CPU Time in Seconds
8K	51.74
16K	29.58
32K	16.34

Observe that the CM-2 CPU time decreases approximately linearly as the number of processors is increased. We note that the same problem when solved on an IBM 3090/600E required 438.35 CPU seconds for the serial Fortran code (cf. Nagurney and Eydeland (1992)), compiled using the FORTVS compiler, optimization level 3, and 291.54 CPU seconds on an IBM 3090/600J. The number of iterations required for convergence was 4 for SEA both on the CM-2 and on the IBM 3090/600. In terms of the parallel runs on the IBM 3090/600E, the wall clock time required for convergence of the parallel implementation of the Splitting Equilibration Algorithm, compiled using the PF compiler, was 444.18 seconds for 1 CPU, 229.85 seconds for 2 CPUs, 118.76 seconds for 4 CPUs, and 86.32 seconds for 6 CPUs.

For additional discussion on the constrained matrix problem and supplementary references, see Nagurney (1993).

4.3 Parallel Computation of Variational Inequality Problems

Here we consider the parallel computation of variational inequality problems (cf. (3.4)). We first discuss the massively parallel computation of spatial price equilibria with ad valorem tariffs via the modified projection method. As mentioned in subsection 3.2.4.1, this algorithm is not a parallel decomposition method per se, but, may, nevertheless, due to the structure of the feasible set K in the application in question, yield subproblems that can be solved simultaneously. This is precisely the feature that is illustrated in this application. We then discuss the parallel computation of multicommodity spatial price equilibrium problems via variational inequality decomposition algorithms and their implementation on a coarse-grained architecture.

4.3.1 Spatial Price Equilibrium Problems with Ad Valorem Tariffs

In this subsection we briefly review the perfectly competitive spatial market model with ad valorem tariffs introduced in Nagurney, Nicholson, and Bishop (1993).

We consider m supply markets involved in the production of a homogeneous commodity and n demand markets. We denote a typical supply market by i and a typical demand market by j. Let s_i denote the supply at supply market i and d_j the demand at demand market j. We group the supplies into a column vector $s \in R^m$ and the demands into a column vector $d \in R^n$. Let Q_{ij} denote the nonnegative commodity shipment between supply and demand market pair (i, j), and group the commodity shipments into a column vector $Q \in R^{mn}$.

The commodity shipments and the supplies and demands must satisfy the following conservation of flow equations:

$$s_i = \sum_{j=1}^{n} Q_{ij}, \quad i = 1, \ldots, m, \tag{4.20}$$

$$d_j = \sum_{i=1}^{m} Q_{ij}, \quad j = 1, \ldots, n. \tag{4.21}$$

Hence, the supply at each supply market must be equal to the sum of the commodity shipments from that market to all the demand markets, and the demand at each demand market must be equal to the sum of the commodity shipments from all supply markets to each demand market.

We now describe the price and cost structure. Let π_i denote the supply price at supply market i and ρ_j the demand price at demand market j. We group the supply prices into a row vector $\pi \in R^m$ and the demand prices into a row vector $\rho \in R^n$. The transportation cost associated with shipping the commodity between supply market i and demand market j is denoted by c_{ij}. We group the transportation costs into a row vector $c \in R^{mn}$.

We assume that the supply price at a supply market may, in general, depend upon the supplies of the commodity at every supply market, that is,

$$\pi = \pi(s). \tag{4.22}$$

Similarly, the demand price at a demand market may, in general, depend upon the demands for the commodity at every demand market, that is,

$$\rho = \rho(d). \tag{4.23}$$

The per unit transportation cost, in turn, associated with shipping the commodity between a pair of supply and demand markets is assumed to be fixed, that is, it is independent of the volume of commodity shipments, where the fixed per unit cost is denoted by $\bar{c}_{ij}$, and the associated mn-dimensional vector by $\bar{c}$. Hence, we have that

$$c = \bar{c}. \tag{4.24}$$

Note that other fixed per unit transfer costs and per unit tariffs can be readily incorporated into the fixed $\bar{c}$ function.

We now introduce discriminatory ad valorem tariffs. Let t_{ij} denote the ad valorem tariff, assumed positive, and applied to imports by demand market j from supply market i. The incorporation of ad valorem tariffs modifies the spatial price equilibrium conditions (cf. Samuelson (1952), Takayama and Judge (1971)) as follows: For all pairs of supply and demand markets (i,j); $i = 1,\ldots,m$; $j = 1,\ldots,n$, a commodity supply, shipment, and demand pattern (s^*, Q^*, d^*) satisfying (4.20) and (4.21) is said to be in equilibrium if

$$(\pi_i(s^*) + \bar{c}_{ij}) \cdot (1 + t_{ij}) \begin{cases} = \rho_j(d^*), & \text{if } Q^*_{ij} > 0 \\ \geq \rho_j(d^*), & \text{if } Q^*_{ij} = 0. \end{cases} \tag{4.25}$$

Hence, in equilibrium, if a positive amount of the commodity is shipped between a pair of supply and demand markets, then the effective supply price plus transportation

cost after the imposition of ad valorem tariffs must be equal to the demand price at the demand market. If there is no commodity shipment between a pair of supply and demand markets, then the effective supply price plus transportation cost can exceed the demand price.

In view of constraints (4.20) and (4.21), one can define the functions $\hat{\pi}_i(Q) \equiv \pi_i(s)$, $i = 1, \ldots, m$, and the functions $\hat{\rho}_j(Q) = \rho_j(d)$, $j = 1, \ldots, n$. The variational inequality formulation of the equilibrium conditions governing the spatial market model with ad valorem tariffs derived in Nagurney, Nicholson, and Bishop (1993) is

$$\sum_{i=1}^{m}\sum_{j=1}^{n}((\hat{\pi}_i(Q^*) + \bar{c}_{ij}) \cdot (1 + t_{ij}) - \hat{\rho}_j(Q^*)) \cdot (Q_{ij} - Q_{ij}^*) \geq 0, \quad \forall Q \in R_+^{mn}. \tag{4.26}$$

If one then defines

$$F_{ij}(Q) \equiv ((\hat{\pi}_i(Q) + \bar{c}_{ij}) \cdot (1 + t_{ij}) - \hat{\rho}_j(Q)), \quad \forall i, j, \tag{4.27}$$

and lets $F(Q) \in R^{mn}$ be the row vector with (i,j)-th component $F_{ij}(Q)$, then variational inequality (4.26) may be expressed in standard form as:

Determine $Q^* \in K$, such that

$$F(Q^*) \cdot (Q - Q^*) \geq 0, \quad \forall Q \in K, \tag{4.28}$$

where the feasible set $K \equiv \{Q | Q \in R_+^{mn}\}$.

The algorithm that we propose is the modified projection method (cf. subsection 3.2.4.1) which resolves the variational inequality problem under consideration here into subproblems that are very simple for computational purposes. Indeed, we obtain a closed form expression for the determination of the commodity shipments at each iteration. Moreover, since each of the commodity shipments between a pair of supply and demand markets can be evaluated separately and simultaneously at any iteration, this algorithmic scheme enables one to exploit the availability of (massively) parallel computer architectures.

We now provide the closed form expressions for the solution of encountered subproblems. In particular, one must first compute: For all supply and demand market pairs (i,j), $i = 1, \ldots, m$; $j = 1, \ldots, n$,

$$\bar{Q}_{ij}^{\tau} = \max\{0, \gamma((-\pi_i(s^{\tau}) - \bar{c}_{ij})(1 + t_{ij}) + \rho_j(d^{\tau})) + Q_{ij}^{\tau}\}, \tag{4.29}$$

and then: For all supply and demand market pairs (i, j), $i = 1, \ldots, m$; $j = 1, \ldots, n$,

$$Q_{ij}^{\tau+1} = \max\{0, \gamma((-\pi_i(\bar{s}^\tau) - \bar{c}_{ij})(1 + t_{ij}) + \rho_j(\bar{d}^\tau)) + Q_{ij}^\tau\}. \tag{4.30}$$

In view of expressions (4.29) and (4.30), one sees that all of the mn commodity shipments can be solved simultaneously at each iteration τ. Hence, an "ideal" computer architecture for the solution of such problems may be one in which there are as many processors as there are pairs of markets. The convergence results can be found in Nagurney, Nicholson, and Bishop (1993).

4.3.1.1 Implementation of the Modified Projection Method on the CM-2

In this subsection some numerical results are presented for the implementations of the modified projection method on two distinct architectures, the IBM ES/9000, when the algorithm is implemented in Fortran, compiled, and executed using a single processor, and the CM-2, when the algorithm is implemented in CM Fortran and executed on 8K, 16K, and 32K processors. We consider the solution of large-scale spatial price equilibrium problems with discriminatory ad valorem tariffs.

Specifically, we consider spatial price equilibrium problems in which the supply price and demand price functions are asymmetric and linear and the transportation cost functions are fixed.

The CM Fortran code for the implementation of the modified projection method for the model consisted of an input and setup routine and a computation routine to implement the iterative steps (4.29) and (4.30). The crucial feature in the design of the program was the construction of the data structures to take advantage of the data level parallelism and computation. We first constructed the array C, of dimension $m \times n$, to store the transportation costs $\{\bar{c}_{ij}\}$. We then constructed the array t to store the tariff rates, with the (i, j)-th component equal to t_{ij}.

The supply price coefficients were stored in an $m \times m$ array SC, and the demand price coefficients were stored in an $n \times n$ array DC. We also introduced additional arrays SP and DP to denote, respectively, the supply prices and the demand prices at a given iteration, where the i-th row of SP consisted of the identical elements $\{\pi_i\}$ and the j-th column of DP consisted of the identical elements $\{\rho_j\}$. To compute the supply prices, we

used the **spread** command to spread the supplies and then multiplied the resulting matrix with the SC matrix. Specifically, the **spread** command makes multiple copies of a vector along columns or along the rows to create a 2-dimensional array. We then used the **sum** command to add the elements of each row of the resulting arrays and added the resulting vector to the vector containing the fixed supply price terms. The result was then **spread** to create the supply prices SP at the particular iteration. The demand prices were obtained in an analogous fashion. The array QO was used to store the values of Q from the previous iteration and was used for convergence purposes.

We now present the critical steps in the CM Fortran computation section.

Implementation of the Modified Projection Method

Do while (err.ge.ϵ)

1. QO(:,:)=Q(:,:)
2. *construct SP and DP*
3. temp(:,:)=Q(:,:)+γ(DP(:,:)-(C(:,:)+SP(:,:))*(1+t))
4. Q(:,:)=temp(:,:)
5. **where**(temp(:,:).lt.0.) Q(:,:)=0.
6. *update SP and DP with new Q*
7. temp(:,:)=QO(:,:)+γ(DP(:,:)-(C(:,:)+SP(:,:))*(1+t))
8. Q(:,:)=temp(:,:)
9. **where**(temp(:,:).lt.0.)Q(:,:)=0.
10. err=**maxval**(abs(Q-QO))
11. *update supplies and demands*

end do

Hence, from step 3 above it can be seen that element (i,j) of the array "temp" contains at the τ-th iteration the value of: $\gamma(\rho_j(d^\tau) - (\bar{c}_{ij} + \pi_i(s^\tau))(1 + t_{ij})) + Q_{ij}^\tau$ (cf. (4.29)). In step 7 above, on the other hand, it can be seen that element (i,j) of the array "temp" now contains at the τ-th iteration the value of: $\gamma(\rho_j(\tilde{d}^\tau) - (\bar{c}_{ij} + \pi_i(\bar{s}^\tau))(1 + t_{ij})) + Q_{ij}^\tau$ (cf. (4.30)). All the variables above followed by a "(:,:)" are 2-dimensional arrays.

Q is updated by using a mask in steps 5 and 9, where the (i,j)-th element is set to zero if the value of temp(i,j) is negative. What is important to note is that, at each

iteration, all of the Q_{ij}'s, for $i = 1, \ldots, m$; $j = 1, \ldots, n$, are computed and updated *simultaneously*. This is not possible when the algorithm is implemented on a serial architecture with consequences that shall be highlighted subsequently.

Note that the above code can be easily adapted to solve spatial price equilibrium problems without ad valorem tariffs, but with fixed unit transportation costs, by simply removing the $(1 + t)$ expression, which we did, as well.

We now turn to the presentation of the numerical results. The problems are large-scale problems ranging in size from one hundred supply markets and one hundred demand markets to five hundred supply markets and five hundred demand markets, that is, with ten thousand to two hundred and fifty thousand commodity shipment variables.

The numerical results for the large-scale problems are reported for both the serial implementation of the algorithm in Fortran on the IBM ES/9000 and the parallel implementation in CM Fortran presented above on the CM-2 architecture.

The tariffs t_{ij} were generated randomly and uniformly in the range: $[0, 2,]$. A full description of the datasets, along with additional numerical results, can be found in Nagurney, Nicholson, and Bishop (1993).

We set the convergence tolerance $\epsilon = .01$, and set $\gamma = .0001$ for all the numerical examples. Also, we initialized the algorithm for each example with $Q_{ij}^0 = 0$ for all i, j.

The serial implementation of the modified projection method on the IBM ES/9000 yielded the same number of iterations as had been obtained on the CM-2 for each example. The CPU times are reported in Table 4.3. We report the times for each example both with the tariffs and with the tariffs removed.

Table 4.3

Numerical Results for Large-Scale Problems with Ad Valorem Tariffs

CPU Times in Seconds

	IBM ES/9000		CM-2 (8K)		CM-2 (16K)		CM-2 (32K)	
Example $m \times n$	Without Tariffs	With Tariffs	Without Tariffs	With Tariffs	Without Tariffs	With Tariffs	Without Tariffs	With Tariffs
100×100	8.80	241.50	7.58	186.72	5.22	130.90	—	—
200×200	53.83	632.08	17.23	178.72	14.96	155.19	10.89	112.89
300×300	107.94	>900	21.28	239.96	14.04	158.19	10.05	113.07
400×400	246.31	>900	38.23	523.15	24.92	340.64	17.43	238.29
500×500	880.47	>900	158.82	657.48	120.78	499.26	68.88	284.58

The first example, 100 × 100, consisting of one hundred supply markets and one hundred demand markets, required 185 iterations for convergence in the absence of tariffs and 4,611 iterations in the presence of tariffs. The second example, 200 × 200, consisting of two hundred supply markets and two hundred demand markets, required 286 iterations for the without-tariff case, and 2,951 iterations for the with-tariff case.

The third example, 300 × 300, consisting of three hundred supply markets and three hundred demand markets, required 250 iterations for convergence for the without-tariff case and 2,796 iterations for the with-tariff case. The fourth example in Table 4.3, 400 × 400, consisting of four hundred supply markets and four hundred demand markets, required 305 iterations for the without-tariff case, and 4,140 iterations for the with-tariff case. The final problem, 500 × 500, consisting of five hundred supply markets and five hundred demand markets, required 686 iterations for convergence for the problem without tariffs, and 2,825 iterations for convergence for the problem with tariffs.

It is apparent that the use of a massively parallel architecture for these large-scale problems realized substantial savings in CPU time over the time required on the serial architecture. For example, in the smallest problem, 100 × 100, and without tariffs, the time on the IBM ES/9000 was 8.8 seconds, whereas the time using 16K processors of the CM-2 was 5.22 seconds. (We did not solve this problem on 32K processors since there were only 10,000 variables in this size of problem.) In the next largest problem, 200 × 200, the

time on the ES/9000 for the problem without tariffs was 53.83, whereas the same problem was solved in only 10.89 seconds using 32K processors of the CM-2, a five-fold improvement. This improvement in relative performance increased as the size of the problem increased, with the result that the largest problem in this set, 500×500, required 880.47 seconds on the ES/9000 and less than a tenth of that time, 68.88 seconds, when 32K processors of the CM-2 were utilized. The largest problem, 500×500, and with tariffs, only required about 4 minuites using 32K processors of the CM-2.

4.3.2 Multicommodity Problems

In this subsection we consider a multicommodity version of the spatial price equilibrium model described in subsection 4.3.1, but without ad valorem tariffs, which will be used as a model for illustrating variational inequality decomposition algorithms. For additional background, see Nagurney (1993).

Consider again m supply markets and n demand markets but now involved the production / consumption of J different commodities, with a typical commodity denoted by k. As before, denote a typical supply market by i and a typical demand market by j. Let s_i^k denote the supply of the commodity k associated with supply market i and let π_i^k denote the supply price of this commodity associated with supply market i. Let d_j^k denote the demand for commodity k associated with demand market j and let ρ_j^k denote the demand price associated with demand market j and commodity k. Group the supplies and supply prices, respectively, into a column vector $s \in R^{Jm}$ and a row vector $\pi \in R^{Jm}$. Similarly, group the demands and the demand prices, respectively, into a column vector $d \in R^{Jn}$ and a row vector $\rho \in R^{Jn}$.

Let Q_{ij}^k denote the nonnegative commodity shipment of commodity k between the supply and demand market pair (i,j) and let c_{ij}^k denote the nonnegative unit transaction cost associated with trading commodity k between (i,j). Assume that the transaction cost includes the cost of transportation; depending upon the application, one may also include a tax/tariff, fee, duty, or subsidy within this cost. Group then the commodity shipments into a column vector $Q \in R^{Jmn}$ and the transaction costs into a row vector $c \in R^{Jmn}$.

The market equilibrium conditions, assuming perfect competition take the following form: For all pairs of supply and demand markets $(i,j) : i = 1,\ldots,m; j = 1,\ldots,n$, and

all commodities $k = 1, \ldots, J$:

$$\pi_i^k(s^*) + c_{ij}^k \begin{cases} = \rho_j^k(d^*), & \text{if} \quad {Q_{ij}^k}^* > 0 \\ \geq \rho_j(d^*), & \text{if} \quad {Q_{ij}^k}^* = 0. \end{cases} \tag{4.31}$$

The condition (4.31) states that if there is trade of commodity k between a market pair (i, j), then the supply price of k at supply market i plus the transaction cost between the pair of markets associated with trading commodity k must be equal to the demand price of k at demand market j in equilibrium; if the supply price plus the transaction cost exceeds the demand price, then there will be no shipment of that commodity between the supply and demand market pair.

Moreover, the following feasibility conditions must hold for every commodity k, and markets i and j:

$$s_i^k = \sum_{j=1}^{n} Q_{ij}^k \tag{4.32}$$

and

$$d_j^k = \sum_{i=1}^{m} Q_{ij}^k. \tag{4.33}$$

The transaction cost between a pair of supply and demand markets associated with trading a commodity may now depend upon the shipments of all the commodities between every pair of markets, that is,

$$c = c(Q) \tag{4.34}$$

where c is a known function.

The variational inequality formulation of the equilibrium conditions (4.31) is

$$\pi(s^*) \cdot (s - s^*) + c(Q^*) \cdot (Q - Q^*) - \rho(d^*) \cdot (d - d^*) \geq 0, \quad \forall (s, Q, d) \in K, \tag{4.35}$$

where $K \equiv \prod_{k=1}^{J} K_k$, where K_k is defined as the set of (s, Q, d), such that constraints (4.32) and (4.33) are satisfied.

The algorithm that was utilized for the solution of this problem was the linear Jacobi method with a diagonal matrix $A(\cdot)$, which resolves the problem into single commodity problems, each of which, in turn, is equivalent to a quadratic programming problem. The algorithm that was utilized for the solution of the embedded subproblems was the

demand market equilibration algorithm of Dafermos and Nagurney (1989). In particular, the algorithm in the context of this application, is expressed as follows.

Linear Gauss-Seidel Method:

Start with an initial feasible $(s^0, Q^0, d^0) \in K$.

At iteration τ, construct new supply price, demand price, and transaction cost functions, which are linear and separable, and given for each commodity k by

$$\pi_i^{\tau,k}(s_i^k) = \frac{\partial \pi_i^k}{\partial s_i^k}(s^\tau)s_i^k + (\pi_i^k(s^\tau) - \frac{\partial \pi_i^k}{\partial s_i^k}(s^\tau)s_i^{\tau,k}), \quad i = 1, \ldots, m,$$

$$\rho_j^{\tau,k}(d_j^k) = \frac{\partial \rho_j^k}{\partial d_j^k}(d^\tau)d_j^k + (\rho_j^k(d^\tau) - \frac{\partial \rho_j^k}{\partial d_j^k}(d^\tau)d_j^{\tau,k}), \quad j = 1, \ldots, n,$$

$$c_{ij}^{\tau,k}(Q_{ij}^k) = \frac{\partial c_{ij}^k}{\partial Q_{ij}^k}(Q^\tau)Q_{ij}^k + (c_{ij}^k(Q^\tau) - \frac{\partial c_{ij}^k}{\partial Q_{ij}^k}(Q^\tau)Q_{ij}^{\tau,k}), \quad i = 1, \ldots, m; j = 1, \ldots, n,$$

and solve the variational inequality subproblem for each k, of the form (4.35), which is equivalent to a quadratic programming problem. The solution is $(s^{\tau+1}, Q^{\tau+1}, d^{\tau+1})$.

In Nagurney and Kim (1989) a problem consisting of 50 supply markets, 50 demand markets, and 12 commodities was solved, where the supply price and demand price functions were quadratic, and the transaction cost functions were highly nonlinear (to the fourth power).

In Table 4.4 the speedups and efficiencies (cf. (2.1) and (2.2)) obtained when the algorithm was implemented on an IBM 3090/600E and compiled using the Parallel Fortran compiler are reported. T_1^* here denotes the time required for the algorithm implemented on a single processor of the system. The task allocation was accomplished by using the constructs provided in Parallel Fortran.

Table 4.4

Speedups and Efficiencies for a Multicommodity Example

N	T_N	T_1^*	S_N	E_N
2	73.34	128.72	1.76	88%
3	55.63	128.72	2.31	77%

Additional numerical results can be found in Nagurney and Kim (1989) for both the linear Jacobi algorithm and the linear Gauss-Seidel algorithm.

4.4 Parallel Computation of Dynamical Systems

In this subsection we consider the computation of dynamical systems via the Euler Method presented in subsection 3.2.5. We first illustrate the method through an application to the classical oligopoly problem and then discuss a massively parallel implementation of the algorithm for the computation of a dynamical systems model of spatial price equilibrium.

4.4.1 Oligopolistic Market Equilibria

In this subsection we first briefly review the oligopoly model and its variational inequality formulation. We then present the dynamical system whose set of stationary points corresponds to the set of solutions of the variational inequality problem.

Assume that there are m firms involved in the production of a homogeneous commodity and a single demand market. Let q_i denote the nonnegative commodity output produced by firm i and let d denote the demand for the commodity at the demand market. Group the production outputs into a column vector $q \in R_+^m$.

The following conservation of flow equation must hold:

$$d = \sum_{i=1}^{m} q_i. \tag{4.36}$$

Associate with each firm i a production cost f_i, where

$$f_i = f_i(q_i). \tag{4.37}$$

The demand price for the commodity is given by

$$p = p(d). \tag{4.38}$$

The profit or utility u_i of firm i is then given by the expression

$$u_i = pq_i - f_i. \tag{4.39}$$

In view of (4.36)-(4.38), one may write the profit as a function solely of the production output, i.e.,

$$u = u(q). \tag{4.40}$$

Now consider the usual oligopolistic market mechanism (cf. Cournot (1838), Nash (1950)), in which the m firms supply the commodity in a noncooperative fashion, each one trying to maximize its own profit. We seek to determine a nonnegative production pattern q^* for which the m firms will be in a state of equilibrium as defined below.

Definition 4.1

A commodity production pattern $q^* \in R^m_+$ is said to constitute a Cournot-Nash equilibrium if for each firm $i; i = 1, \ldots, m$,

$$u_i(q_i^*, \hat{q_i^*}) \geq u_i(q_i, \hat{q_i^*}), \quad \forall q_i \in R_+, \tag{4.41}$$

where $\hat{q_i^*} \equiv (q_1^*, \ldots, q_{i-1}^*, q_{i+1}^*, \ldots, q_m^*)$.

As established in Gabay and Moulin (1980), the variational inequality formulation of the Cournot-Nash equilibrium is as follows.

Assume that for each firm i the profit function $u_i(q)$ is concave with respect to the variables $\{q_1, \ldots, q_m\}$, and continuously differentiable. Then $q^* \in R^m_+$ is a Cournot-Nash equilibrium if and only if it satisfies the variational inequality

$$-\sum_{i=1}^{m} \frac{\partial u_i(q^*)}{\partial q_i} \cdot (q_i - q_i^*) \geq 0, \quad \forall q \in R^m_+, \tag{4.42}$$

or, equivalently, q^* is an equilibrium production pattern if and only if it satisfies the variational inequality

$$\sum_{i=1}^{m} \left[\frac{\partial f_i(q_i^*)}{\partial q_i} - \frac{\partial p(\sum_{i=1}^{m} q_i^*)}{\partial q_i} q_i^* - p(\sum_{i=1}^{m} q_i^*) \right] \times [q_i - q_i^*] \geq 0, \quad \forall q \in R^m_+. \tag{4.43}$$

We will now put the oligopolistic market equilibrium problem into standard variational inequality form. Let x be the column vector $x \equiv q \in R^m$, and let $F(x) \in R^m$ be the row vector with components: $(-\frac{\partial u_1(q_1)}{\partial q_1}, \ldots, -\frac{\partial u_m(q_m)}{\partial q_m})$, and $K \equiv \{q | q \geq 0\}$, then variational inequality (4.43) governing the classical Cournot-Nash oligopoly problem can be placed in standard form.

We now state the ordinary differential equation (ODE) (cf. (3.5) and (3.6)). The class of pertinent ODEs takes the form:

$$\dot{x} = \Pi(x, -F(x)), \quad x(0) = x_0 \in K. \tag{4.44}$$

As established in Lemma 1 in Dupuis and Nagurney (1993), each stationary point of (4.44), that is, each point in the set of x^* satisfying

$$0 = \Pi(x^*, -F(x^*)), \tag{4.45}$$

also satisfies the variational inequality (4.42).

The ordinary differential equation (4.44), however, is not standard in that the right-hand side is discontinuous. Nevertheless, as has been established in Dupuis and Nagurney (1993), the important qualitative and quantitative results of "standard" ODEs will still be applicable.

We now briefly interpret the ODE (4.44) in the context of the oligopoly model. First, note that ODE (4.44) ensures that the production outputs are always nonnegative. Indeed, if one were to consider, instead, the ordinary differential equation: $\dot{x} = -F(x)$, such an ODE would not ensure that $x(t) \geq 0$ for all $t \geq 0$, unless additional restrictive assumptions were to be imposed, such as the assumption that the solutions to the oligopoly problems lie in the interior of the feasible set (cf. Okuguchi (1976) and Okuguchi and Szidarovsky (1990)). ODE (4.44), however, retains the interpretation that if x at time t lies in the interior of K, then the rate at which x changes is greatest when the vector field $-F(x)$ is greatest. Moreover, when the vector field pushes x to the boundary of the feasible set K, then the projection Π ensures that x stays within K.

Recall now the definition of $F(x)$ for the oligopoly model, in which case the dynamical system (4.44) states that the rate of change of the production outputs is greatest when the firms' marginal utilities are greatest. If the marginal utilities are positive, then the firms will increase their shipments; if they are negative, then they will decrease their shipments. Therefore, ODE (4.44) is a continuous adjustment or tatonnement process for the oligopoly problem. Although the dynamical system provides a continuous adjustment process, a discrete time process is needed for actual computational purposes. In particular, in the context of the classical oligopoly model, one would, at each iteration τ of the Euler Method, compute the new production outputs for each firm i in closed form as follows

$$q_i^{\tau+1} = \max\{0, a_\tau \frac{\partial u_i(q_i^\tau)}{\partial q_i} + q_i^\tau\}, \quad \text{for each} \quad i = 1, \ldots, m. \tag{4.46}$$

Observe that (4.46) is a parallel adjustment process, where in the classical oligopoly problem all of the production outputs are updated simultaneously. Proof of convergence

of the Euler Method for this model, as well as for a spatial oligopoly model, can be found in Nagurney, Dupuis, and Zhang (1993).

It is worth noting the similarity between the Euler-type method and the Goldstein-Levitin-Polyak Gradient Projection Method, cf. Goldstein (1964, 1967) and Levitin and Polyak (1966), who independently proposed a projection method for minimizing a continuously differentiable function $f : K \mapsto R$, where the iteration τ takes the form: $x_\tau = P(x_\tau - a_\tau \nabla f(x_\tau))$ (see also Bertsekas (1976)). The oligopoly problem, however, is a variational inequality problem and not an optimization problem, and although the Euler-Type Method can be used to solve an optimization problem, the converse does not hold true, that is, an optimization algorithm cannot be used to solve a variational inequality problem (unless it can also be cast as an optimization problem, which would hold in the very special case where the Jacobian of F is symmetric).

It is also worth mentioning that Arrow and Hurwicz (1958a,b) earlier proposed a gradient method for optimization problems, which was stated as solving a dynamical system. A discussion of other gradient-type methods, based on both the Hildreth (1957) and the Arrow-Hurwicz methods, and their application to classical spatial price equilibrium problems, can be found in Takayama and Judge (1971).

4.4.1.1 A Numerical Example

We now apply the Euler-Type Method to compute the solution to a numerical example. The algorithm was coded in Fortran and the system used for the numerical work was the IBM ES/9000.

The example is taken from Murphy, Sherali, and Soyster (1982). The oligopoly consists of five firms, each with a production cost function of the form:

$$f_i(q_i) = c_i q_i + \frac{\beta_i}{(\beta_i + 1)} h_i^{-\frac{1}{\beta_i}} q_i^{\frac{(\beta_i+1)}{\beta_i}}, \tag{4.47}$$

with the parameters given in Table 4.5. The demand price function is given by:

$$p(\sum_{i=1}^{5} q_i) = 5000^{\frac{1}{1.1}} (\sum_{i=1}^{5} q_i)^{-\frac{1}{1.1}}. \tag{4.48}$$

Table 4.5

Parameters for the 5-Firm Oligopoly Example

firm i	c_i	h_i	β_i
1	10	5	1.2
2	8	5	1.1
3	6	5	1.0
4	4	5	.9
5	2	5	.8

The convergence criterion was: $|q_i^\tau - q_i^{\tau-1}| \leq .001$, for all i. The algorithm was initialized at $q^0 = (10, 10, 10, 10, 10)$. We utilized the sequence: $\{a_\tau\}$=$10 \times \{1, \frac{1}{2}, \frac{1}{2}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \ldots\}$.

The algorithm required 19 iterations and only a negligible amount of CPU time for convergence. The algorithm converged to q^*=(36.93, 41.81, 43.70, 42.65, 39.17), reported to four digits of accuracy.

As reported in Nagurney (1993), the projection method, which would in the above general iterative scheme (cf. (3.9)) correspond to $F_\tau(x_\tau) = F(x_\tau)$ with $a_\tau = \gamma$, for all iterations τ, required 33 iterations for convergence to the same solution with $\gamma = .9$, under the same initial conditions. The relaxation method, on the other hand, cf. Nagurney (1993), required only 23 iterations but was more computationally costly, since at each iteration nonlinear equations must be solved. Also, we emphasize that the conditions for convergence of both the projection and the relaxation method are more restrictive than those required by the general iterative scheme described in subsection 3.2.5.

4.4.2 Spatial Price Equilibria

In this subsection we consider a dynamical systems model of a single commodity version of the spatial price equilibrium model described in 4.3.2. For additional background and numerical results, see Nagurney, Takayama, and Zhang (1993).

We do, however, use an alternative variational inequality formulation, which makes the massively parallel decomposition by market pairs more apparent. The decomposition proposed here is of the finest possible for this problem. In particular, we consider the variational inequality formulation of the problem given by

$$F(Q^*)^T \cdot (Q - Q^*) \geq 0, \quad \forall Q \in K, \tag{4.49}$$

here $F(\cdot)$ is the mn-dimensional row vector whose (i,j)-th component is given by: $\pi_i(s) + c_{ij}(Q) - \rho_j(d)$, and the feasible set K is defined as the nonnegative orthant: $K \equiv \{Q | Q \geq 0\}$.

We now present the ordinary differential equation (ODE), whose set of stationary points corresponds to the set of solutions of variational inequality (4.49), or, equivalently, to the set of spatial price equilibrium patterns satisfying conditions (4.31), with the number of commodities $J = 1$. The pertinent ODE is given by:

$$\dot{Q} = \Pi(Q, -F(Q)), \quad Q(0) = Q^0 \in K. \tag{4.50}$$

The intuition behind the dynamical system in the context of the spatial price equilibrium problem will now be briefly addressed. If $Q(t) \in K^0$, that is, in the context of the spatial price equilibrium problem, all the commodity shipments at time t, $Q(t)$, are positive, then the evolution of the solution is directly given in terms of $F : \dot{Q} = -F(Q)$, where recall that $-F_{ij}(Q) = \rho_j(d) - c_{ij}(Q) - \pi_i(s)$. In other words, if the demand price at a demand market exceeds the supply price plus transaction cost associated with shipping the commodity between this pair of supply and demand markets, then the commodity shipment between this pair of markets will increase. On the other hand, if the supply price plus transaction cost exceeds the demand price, then the commodity shipment between the pair of supply and demand markets will decrease. If a stationary point is reached, that is, if $\dot{Q} = 0 = -F(Q)$, then the supply price plus the transaction cost will be exactly equal to the demand price for each pair of markets and the associated commodity shipments will be positive.

However, if the vector field F drives Q to the boundary of K, (i.e. $F(Q(t))$ points "out" of K) the right-hand side of (4.50) becomes the projection of F onto ∂K. In other words, if the commodity shipment is driven to be negative, then the projection ensures that the commodity will be nonnegative, by setting it equal to zero.

For the computation of the solution to this problem, we applied the Euler Method (cf. Section 3.2.5), where the expression (3.10) now takes the form:

$$Q_{ij}^{\tau+1} = \max\{0, a_\tau(\rho_j(d^\tau) - c_{ij}(Q^\tau) - \pi_i(s^\tau)) + Q_{ij}^\tau\}, \quad i = 1, \ldots, m; j = 1, \ldots, n. \tag{4.51}$$

Note that (4.51) is a parallel adjustment process in that each of the mn market

pair subproblems can be solved simultaneously at each iteration. Moreover, each such subproblem can be solved explicitly in closed form.

In view of the similarity between the iterative step (4.51) and the iterative steps (4.29) and (4.30), the massively parallel implementation of the Euler Method in CM Fortran is similar to the massively parallel implementation of a single step of the Modified Projection Method.

For completeness, and easy reference, we present some numerical results. In particular, we considered spatial price equilibrium problems with linear, asymmetric supply price functions, linear, asymmetric demand price functions, and quadratic transaction (transportation) cost functions. We report the results on a set of five examples, the first example consisting of 100 supply markets and 100 demand markets, with 10,000 variables or unknown commodity shipments, and ending with a problem with 500 supply markets and 500 demand markets, that is, with 250,000 variables. The numerical results are reported in Table 4.6 for the examples using 8K, 16K, and, finally, 32K processors of the CM-2.

The algorithm was initialized with $Q^0 = 0$. The convergence criterion used was: $|Q_{ij}^{\tau+1} - Q_{ij}^{\tau}| \leq \epsilon$ for all i, j, with the tolerance ϵ set to .001.

Each example (except for the first, which had only 10,000 variables) was solved with 8K processors, with 16K processors, and, finally, with 32K processors.

Table 4.6

CM-2 Times for Spatial Price Equilibrium Problems

Nonlinear Transportation Costs

	# of Supply	# of Demand		CM-2 Time (sec.)	
Example	Markets	Markets	8K	16K	32K
ASP100	100	100	48.98	37.27	—
ASP200	200	200	165.70	154.19	111.88
ASP300	300	300	263.69	172.95	122.80
ASP400	400	400	544.84	352.61	245.17
ASP500	500	500	1772.58	1214.51	690.65

The first example in this set, ASP100, required 2,558 iterations for convergence, the second example, ASP200, required 5,693 iterations, the third example, ASP300, took 5,869 iterations, the fourth example, ASP400, took 8,188 iterations, and the fifth example,

ASP500, 13,264 iterations.

We then considered the solution of spatial price equilibrium problems in which the transportation cost functions were fixed, that is, of the form (4.29) and applied the Euler Method on the CM-2, the CM-5, and the ES/9000. These problems has been previously solved on the CM-2 in Nagurney, Takayama, and Zhang (1993). The problems ranged in size from 300 × 300 or 90,000 variables for SP300 to 500 × 500 or 250,000 variables for SP500 and the CPU times on the three distinct architectures are reported in Table 4.7.

Table 4.7

CM-2, CM-5, and ES/9000 CPU Times for Spatial Price Equilibrium Problems

Fixed Transportation Costs

CPU Times in Seconds

	CM-2		CM-5		ES/9000
Example	32K	128	256	512	1
SP300	93.31	54.31	45.68	40.92	1,170.59
SP400	243.33	311.54	106.49	90.31	4,034.25
SP500	686.78	305.88	180.38	133.80	9,600*

Table 4.7 reports the CPU times using 32K processors of the CM-2, and 128 nodes, 256 nodes, and 512 nodes of the CM-5. Only a single processor of the ES/9000 is also used. The examples (as one would expect) required the same number of iterations for convergence on the CM-5 as they did, respectively, on the CM-2 and on the ES/9000. SP300 required 4,483 iterations, SP400 required 8,187 iterations, whereas SP500 required 13,262 iterations. The CPU time on the ES/9000 for SP500 is estimated, since it became prohibitively expensive to solve it serially.

The numerical results clearly indicate the following. First, it is imperative that an algorithm be mapped to the appropriate architecture. The Euler Method in its realization in the economic equilibrium problem under consideration here is a massively parallel algorithm and, hence, should be implemented on a massively parallel architecture. Indeed, although the Euler mMthod requires many iterations for convergence, the total time required for convergence is minimal in the massively parallel implementation for spatial price equilibrium problems, since each iteration is computationally inexpensive, because of its simplicity and because the problems are solved *simultaneously*.

Second, the ease of portability of the CM Fortran code between the CM-2 and CM-5 was demonstrated. No changes to the code were needed (except for the compilation) in order to execute the CM Fortran code on the CM-5 which had been developed for the CM-2. Third, the numerical results on the CM-5 suggest that very large-scale problems in economics can be solved very efficiently. Indeed, the largest problem, consisting of 250,000 variables required only approximately 2 minutes for solution. This is due, partially, to the fact of the layout of the data structures and partially to the algorithm itself and its implementation in CM Fortran.

Moreover, these results and those in the preceding numerical subsections suggest that massively parallel computation can enable one to conduct many simulations of alternative policy interventions such as, for example, different tariff structures, in a timely fashion.

Acknowledgments

The research reported herein was supported, in part, by the National Science Foundation under grant DMS 9024071 under the Faculty Awards for Women program.

This research was conducted at the National Center for Supercomputer Applications at the University of Illinois at Urbana-Champaign, at the Pittsburgh Supercomputing Center, and at the Cornell Theory Center at Cornell University in Ithaca, New York. The use of these facilities and the technical assistance provided at these centers are gratefully acknowledged.

The author would also like to thank the computer scientists Marilynn Livingston and D. R. Mani for many helpful discussions in the course of preparing this work and Kathy Dhanda for assistance with the literature searches.

References

Amdahl, G., "The validity of single processor approach to achieving large scale computing capabilities," in **AFIPS Proceedings**, 1967, pp. 483-485.

Amman, H. M., "Applying the Cyber 205 for optimal control experiments in economics," *Supercomputer* **8/9** (1985) 71-74.

Amman, H. M., "Nonlinear control simulation on a vector machine," *Parallel Computing* **10** (1989) 123-127.

Ando, A., Beaumont, P., and Ando, M., "Efficiency of the CYBER 205 for stochastic simulations of a simultaneous, nonlinear, dynamic econometric model," *The International Journal of Supercomputer Applications* **1** (1987) 54-81.

Arrow, K. J., and Hurwicz, L., "Gradient method for concave programming, I: local results," in **Studies in Linear and Nonlinear Programming**, K. J. Arrow, L. Hurwicz, and H. Uzawa, eds., Stanford University Press, Stanford, California, 1958a, pp. 117-126.

Arrow, K. J., and Hurwicz, L., "Gradient method for concave programming, III: further global results and applications to resource allocation," in **Studies in Linear and Nonlinear Programming**, K. J. Arrow, L. Hurwicz, and H. Uzawa, eds., Stanford University Press, Stanford, California, 1958b, pp. 133-145.

Arrow, K. J. , Hurwicz, L., and Uzawa, H., eds., **Studies in Linear and Nonlinear Programming**, Stanford University Press, Stanford, California, 1958.

Bacharach, M., **Biproportional Scaling and Input-Output Change**, Cambridge University Press, Cambridge, United Kingdom, 1970.

Bertsekas, D. P., "On the Goldstein-Levitin-Polyak gradient projection method," *IEEE Transactions on Automatic Control* **AC-21** (1976) 174-184.

Bertsekas, D. P., and Tsitsiklis, J. N., **Parallel and Distributed Computation**, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

Casti, J., Richardson, M., and Larson, R., "Dynamic programming and parallel computers," *Journal of Optimization Theory and Applications* **12** (1973) 423-438.

Chow, G. C., **Analysis and Control of Dynamical Systems**, John Wiley & Sons, New York, 1975.

Coddington, E., and Levinson, N., **Theory of Differential Equations**, McGraw-Hill, New York, New York, 1955.

Cournot, A., **Researches into Mathematical Principles of the Theory of Wealth**, 1838, English translation, McMillan, New York, New York, 1987.

Cray Research, Inc., "CRAY X-MP Computer Systems Functional Description Manual," Eagan, Minnesota, 1986.

Cray Research, Inc., "CRAY C90 Series Functional Description Manual," Eagan, Minnesota, 1993a.

Cray Research, Inc., "Cray T3D System Architecture Overview Manual," Eagan, Minnesota, 1993b.

Dafermos, S., "An iterative scheme for variational inequalities," *Mathematical Programming* **16** (1983) 40-47.

Dafermos, S., "Isomorphic multiclass spatial price and multimodal traffic network equilibrium models," *Regional Science and Urban Economics* **16** (1986) 197-209.

Dafermos, S., and Nagurney, A., "Supply and demand equilibration algorithms for a class of market equilibrium problems," *Transportation Science* **23** (1989) 118-124.

Dafermos, S., and Sparrow, F. T., "The traffic assignment problem for a general network," *Journal of Research of the National Bureau of Standards* **73B** (1969) 91-118.

Dantzig, G. B., "A proof of the equivalence of the programming problem and the game problem," in **Activity Analysis of Producton and Allocation**, T. C. Koopmans, ed., John Wiley, New York, New York, 1951, pp. 330-335.

DeCegama, A. L., **The Technology of Parallel Processing: Parallel Processing Architectures and VLSI Hardware**, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

Deng, Y., Glimm, J., and Sharp, D. H., "Perspectives on parallel computing," *Daedalus* (1992) 31-52.

Denning, P. J., and Tichy, W. F., "Highly parallel computation," *Science* **250**, November 30 (1990) 1217-1222.

Dennis, J. E., and Schnabel, R. B., **Numerical Methods for Unconstrained Optimization and Nonlinear Equations**, Prentice- Hall, Inc., Englewood Cliffs, New Jersey,

1983.

Dixon, P. B., Bowles, S., and Kendrick, D., **Notes and Problems in Microeconomic Theory**, North-Holland, Amsterdam, The Netherlands, 1980.

Dorfman, R., Samuelson, P. A., and Solow, R., **Linear Programming and Economic Analysis**, McGraw-Hill Book Company, New York, New York, 1958.

Dupuis, P., and Nagurney, A., "Dynamical systems and variational inequalities," *Annals of Operations Research* **44** (1993) 9-42.

Eydeland, A., and Nagurney, A., "Progressive equilibration algorithms: the case of linear transaction costs," *Computer Science in Economics and Management* **2** (1989) 197-219.

Finkel, R., and Manber, U., "DIB - A distributed implementation of backtracking," *ACM Trans. Prog. Lang. Syst.* **9** (1987) 235-256.

Flynn, M. J., "Some computer organizations and their effectiveness," **IEEE Transactions on Computers, C-21** (1972) 948-960.

Gabay, D., and Moulin, H., "On the uniqueness and stability of Nash equilibria in noncooperative games," in **Applied Stochastic Control in Econometrics and Management Science**, A. Bensoussan, P. Kleindorfer, and C. S. Tapiero, eds., North-Holland, Amsterdam, The Netherlands, 1980, pp. 271-294.

Garcia, C. B., and Zangwill, W. I., **Pathways to Solutions, Fixed Points, and Equilibria**, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.

Gilli, M., and Pauletto, G., "Econometric model simulation on parallel computers," *The International Journal of Supercomputer Applications* **7.3** (1993).

Goffe, W., Ferrier, G. D., and Rogers, J., "Global optimization of statistical functions with simulated annealing," *Journal of Econometrics*, 1993, in press.

Goldstein, A. A., "Convex programming in Hilbert space," *Bulletin of the Mathematical Society* **70** (1964) 709-710.

Goldstein, A. A., **Constructive Real Analysis**, Harper & Row, New York, New York, 1967.

Hartman, P., **Ordinary Differential Equations**, John Wiley & Sons, New York, New York, 1964.

Hartman, P., and Stampacchia, G., "On some nonlinear elliptical differential functional equations," *Acta Mathematica* **115** (1966) 271-310.

Hennessy, J. L., and Patterson, D. A., **Computer Architecture: A Quantitative Approach**, Morgan Kaufmann, San Mateo, California, 1990.

High Performance Fortran Forum, "High Performance Fortran Language Specification, version 4," Rice University, Houston, Texas, 1992.

Hildreth, C., "A quadratic programming procedure," *Naval Research Logistics Quarterly* **4** (1957) 79-85.

Hillis, W. D., "What is massively parallel computing, and why is it important?" *Daedalus* (1992) 1-15.

Hockney, R., and Jesshope, C., **Parallel Computers**, Adam Hilger Ltd., Bristol, England, 1981.

Holbrook, R. S., "A practical method for controlling a large, nonlinear, stochastic system," *Annals of Economic and Social Measurement* **3** (1974) 155-176.

Hughes, M., and Nagurney, A., "A network model and algorithm for the analysis and estimation of financial flow of funds," *Computer Science in Economics and Management* **5** (1992) 23-39.

IBM Corporation, "Parallel FORTRAN Language and Library Reference," Document Number SC23-0431-0, 1988.

IBM Corporation, "IBM Enterprise System/9000: Introducing the System," GA24-4186, Endicott, New York, 1992.

IBM Corporation, "IBM 9076 Scalable POWERparallel Systems: General Information," GH26-7219, Kingston, New York, 1993.

Intel Corporation, "Paragon Supercomputers," Beaverton, Oregon, 1992.

Intriligator, M. D., **Mathematical Optimization and Economic Theory**, Prentice-Hall, Englewood Cliffs, New Jersey, 1971.

Judd, K. L., **Numerical Methods in Economics**, Hoover Institution, Stanford University, Stanford, California, 1991.

Kantorovich, L. V., **The Best Use of Economic Resources**, Moscow Akademia Nauk, 1959, English translation, Harvard University Press, Cambridge, Massachusetts, 1965.

Kaufmann III, W. J., and Smarr, L. L., **Supercomputing and the Transformation of Science**, Scientific American Library, New York, New York, 1993.

Kendall Square Research, "KSR1 Technical Summary," Waltham, Massachusetts, 1992.

Kendrick, D. A., "Stochastic control in macroeconomic models," IEEE conference publication no. 101, London, England, 1973.

Kim, D. S., and Nagurney, A., "Massively parallel implementation of the Splitting Equilibration Algorithm," *Computational Economics* (1993), in press.

Kinderlehrer, D., and Stampacchia, D., **An Introduction to Variational Inequalities and Their Applications**, Academic Press, New York, New York, 1980.

Kindervater, G. A., and Lenstra, J. K., "Parallel computing in combinatorial optimization," *Annals of Operations Research* **14** (1988) 245-289.

Koopmans, T. C., **Activity Analysis of Production and Allocation**, John Wiley & Sons, New York, New York, 1951.

Korpelevich, G. M., "The extragradient method for finding saddle points and other problems," *Ekonomicheskie i Mathematicheskie Metody*, translated as *Matekon* **12** (1976) 747-756.

Lasdon, L. S., **Optimization Theory for Large Systems**, Macmillan, New York, New York, 1970.

Leighton, F. T., **Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes**, Kaufmann Publishers, San Mateo, California, 1992.

Lemke, C. E., "A survey of complementarity problems," in **Variational Inequalities and Complementarity Problems**, R. W. Cottle, F. Giannessi, and J. L. Lions, eds., John Wiley & Sons, Chichester, England, 1980, pp. 213-239.

Lootsma, F. A., and Ragsdell, K. M., "State-of-the-art in parallel nonlinear optimization," *Parallel Computing* **6** (1988) 133-155.

Markowitz, H., "Portfolio selection," *The Journal of Finance* **7** (1952) 77-91.

Markowitz, H., **Portfolio Selection: Efficient Diversification of Investments**, John Wiley & Sons, New York, New York, 1959.

Murphy, F. H., Sherali, H. D., and Soyster, A. L., "A mathematical programming approach for determining oligopolistic market equilibrium," *Mathematical Programming* **24** (1982) 92-106.

Nagurney, A., **Network Economics: A Variational Inequality Approach**, Kluwer Academic Publishers, Boston, MA, 1993.

Nagurney, A., "Variational inequalities in the analysis and computation of multi-sector, multi-instrument financial equilibria," *Journal of Economic Dynamics and Control* **18** (1994) 161-184.

Nagurney, A., Dong, J., and Hughes, M., "The formulation and computation of general financial equilibrium," *Optimization* **26** (1992) 339-354.

Nagurney, A., Dupuis, P., and Zhang, D., "A dynamical systems approach for network oligopolies and variational inequalities," School of Management, University of Massachusetts, Amherst, Massachusetts, 1992.

Nagurney, A., and Eydeland, A., "A Splitting Equilibration Algorithm for the computation of large-scale constrained matrix problems: theoretical analysis and applications," **Computational Economics and Econometrics**, *Advanced Studies in Theoretical and Applied Econometrics* **22**, H. M. Amman, D. A. Belsley, and L. F. Pau, eds., 1992, pp. 65-105.

Nagurney, A., and Hughes, M., "Financial flow of funds networks," *Networks* **22** (1992) 145-161.

Nagurney, A., and Kim, D. S., "Parallel and serial variational inequality decomposition algorithms for multicommodity market equilibrium problems," *The International Journal of Supercomputer Applications* **3** (1989) 34-58.

Nagurney, A., Nicholson, C. F., and Bishop, P. M., "Spatial price equilibrium models with discriminatory ad valorem tariffs: formulation and comparative computation using variational inequalities," School of Management, University of Massachusetts, Amherst, Massachusetts, 1993.

Nagurney, A., Takayama, T., and Zhang, D., "Massively parallel computation of spatial price equilibria as dynamical systems," 1993, to appear in *Journal of Economic Dynamics and Control.*

Nash, J. F., "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences* **36** (1950) 48-49.

Norman, A. L., "First order dual control," *Annals of Economic and Social Measurement* **5** (1976) 311-322.

Okuguchi, K., **Expectations and Stability in Oligopoly Models**, *Lecture Notes in Economics and Mathematical Systems* **138**, Springer-Verlag, Berlin, Germany, 1976.

Okuguchi, K., and Szidarovsky, F., **The Theory of Oligopoly with Multi-Product Firms**, *Lecture Notes in Economics and Mathematical Systems* **342**, Springer-Verlag, Berlin, Germany, 1990.

Ortega, J. M., and Voigt, R. G., "Solution of partial differential equations on vector and parallel computers," *SIAM Review* **27** (1985) 159-240.

Pardalos, P. M., and Rosen, J. B., **Constrained Global Optimization: Algorithms and Applications**, *Lecture Notes in Computer Science* **268**, Springer-Verlag, Berlin, Germany, 1987.

Petersen, C. E., "Computer simulation of large-scale econometric models: Project LINK," *The International Journal of Supercomputer Applications* **1** (1987) 31-54.

Petersen, C. E., and Cividini, A., "Vectorization and econometric model simulation," *Computer Science in Economics and Management* **2** (1989) 103-117.

Ralston, A., and Reilly, E. D., eds., **Encyclopedia of Computer Science**, third edition, Van Nostrand Reinhold, New York, New York, 1993.

Samuelson, P. A., "A spatial price equilibrium and linear programming," *American Economic Review* **42** (1952) 283-303.

Scarf, H., "The approximation of fixed points of continuous mappings," *SIAM Journal of Applied Mathematics* **15** (1964) 1328-1343.

Scarf, H., with T. E. Hansen, **The Computation of Economic Equilibria**, Yale University Press, New Haven, Connecticut, 1973.

Sharpe, W., **Portfolio Theory and Capital Markets**, McGraw-Hill Book Company, New York, New York, 1970.

Smale, S., "A convergent process of price adjustment and Global Newton methods," *Journal of Mathematical Economics* **3** (1976) 1-14.

Steele, Jr., G. L., "Languages for massively parallel computers," in **Proceedings of the IEEE Second Symposium on the Frontiers of Massively Parallel Computations**, 1988, pp. 3-13.

Takayama, A., **Mathematical Economics**, Dryden Press, Hillsdale, New Jersey, 1974.

Takayama, T., and Judge, G. G., "Equilibrium among spatially separated markets: a reformulation," *Econometrica* **32** (1964) 510-524.

Takayama, T., and Judge, G. G., **Spatial and Temporal Price and Allocation Models**, North-Holland, Amsterdam, The Netherlands, 1971.

Thinking Machines Corporation, "CM-2 Technical Summary," Cambridge, Massachusetts, 1990.

Thinking Machines Corporation, "CM-5 Technical Summary," Cambridge, Massachusetts, 1992a.

Thinking Machines Corporation, "Getting Started in CM Fortran," Cambridge, Massachusetts, 1992b.

Thinking Machines Corporation, "CMSSL Release Notes for the CM-200," Cambridge, Massachusetts, 1992c.

Thinking Machines Corporation, "CM Fortran User's Guide," Cambridge, Massachusetts, 1993a.

Thinking Machines Corporation, "CMMD User's Guide," Cambridge, Massachusetts, 1993b.

Thinking Machines Corporation, "Using the CMAX Converter," Cambridge, Massachusetts, 1993c.

Varian, H., "Dynamical systems with applications to economics," in **Handbook of Mathematical Economics**, K. J. Arrow and M. D. Intriligator, eds., 1981, pp. 93-110.