NBER WORKING PAPER SERIES

SOLVING DYNAMIC PROGRAMMING PROBLEMS ON A COMPUTATIONAL GRID

Yongyang Cai
Kenneth L. Judd
Greg Thain
Stephen J. Wright

Working Paper 18714
http://www.nber.org/papers/w18714

NATIONAL BUREAU OF ECONOMIC RESEARCH
1050 Massachusetts Avenue
Cambridge, MA 02138
January 2013

Cai and Judd gratefully acknowledge National Science Foundation support (SES- 0951576). We also thank Miron Livny for his generous support and access to the HTCondor cluster at the University of Wisconsin-Madison. The views expressed herein are those of the authors and do not necessarily reflect the views of the National Bureau of Economic Research.

Solving Dynamic Programming Problems on a Computational Grid
Yongyang Cai, Kenneth L. Judd, Greg Thain, and Stephen J. Wright
NBER Working Paper No. 18714
January 2013
JEL No. C61,C63,G11

ABSTRACT

We implement a dynamic programming algorithm on a computational grid consisting of loosely coupled processors, possibly including clusters and individual workstations. The grid changes dynamically during the computation, as processors enter and leave the pool of workstations. The algorithm is implemented using the Master-Worker library running on the HTCondor grid computing platform. We implement value function iteration for several large dynamic programming problems of two kinds: optimal growth problems and dynamic portfolio problems. We present examples that solve in hours on HTCondor but would take weeks if executed on a single workstation. The use of HTCondor can increase a researcher's computational productivity by at least two orders of magnitude.

Yongyang Cai
Hoover Institution
Stanford University
Stanford, CA 94305
yycai@stanford.edu

Kenneth L. Judd
Hoover Institution
Stanford University
Stanford, CA 94305-6010
and NBER
kennethjudd@mac.com

Greg Thain
Computer Science Department
University of Wisconsin-Madison
WI 53706, USA
gthain@cs.wisc.edu

Stephen J. Wright
Computer Science Department
University of Wisconsin-Madison
WI 53706, USA
swright@cs.wisc.edu

1 Introduction: Motivation and Model

Many economic optimization problems require weeks or months of CPU time, or even more, to solve because of the "curse of dimensionality". Parallelization is a natural approach to break the "curse of dimensionality" because it allows you to use hundreds of hours of CPU time within one wall clock hour if you have hundreds of CPUs working together. This paper uses a user-friendly parallelization tool, Master-Worker (MW), on HTCondor to show that dynamic programming problems can fully utilize the potential value of parallelism on hardware available to most economists. It also is one of the first large uses of parallel computation in dynamic programming.

Dynamic programming (DP) is the essential tool in solving problems of dynamic and stochastic controls in economic analysis. Many DP problems are solved by value function iteration, where the period t value function is computed from the period $t+1$ value function, and the value function is known at the terminal time T. A set of discrete and approximation nodes will be chosen and the period t value function at those nodes will be computed and then we can use some approximation methods to approximate the value function. For every approximation node, there is a time-consuming optimization problem to be solved. Moreover, these optimization problems are independent, allowing them to be solved efficiently in parallel.

This paper is constructed as follows. Section 2 gives an introduction of HTCondor-MW system. Section 3 describes DP algorithms. Section 4 introduces two types of parallel DP algorithms in the HTCondor-MW system. Section 5 and 6, respectively, give computational results of the parallel DP algorithms in the HTCondor-MW system for solving multidimensional optimal growth problems and dynamic portfolio optimization problems.

2 A Grid Platform

The HTCondor system is a high-throughput computing (HTC), open-source software framework for distributed parallelization of computationally intensive tasks on a cluster of computers. The HTCondor software is freely available to all; see http://research.cs.wisc.edu/htcondor/index.html for details. HTCondor acts as a management tool for identifying, allocating and managing available resources to solve large distributed computations. For

example, if a workstation on a network is currently unused, HTCondor will detect that fact, and send it a task. HTCondor will continue to use that workstation until a higher-priority user (such as a student sitting at the keyboard) appears, at which time HTCondor ends its use of the workstation. This is called "cycle scavenging" and allows a system to take advantage of essentially free computing time. HTCondor can also be used on a dedicated cluster.

The HTCondor team at the University of Wisconsin-Madison has developed several "flavors" of HTCondor, each fine-tuned for some specific type of parallel programming. In this paper we use the HTCondor Master-Worker (MW) system for parallel algorithms to solve DP problems. The HTCondor MW system consists of two entities: a master process and a cluster of worker processes. The master process decomposes the problem into small tasks and puts those tasks in a queue. Each worker process first examines the queue, takes the "top" problem off the queue and solves it. The worker then sends the results to the master, examines the queue of unfinished tasks, and repeats this process until the queue is empty. The workers' execution is a simple cycle: take a task off master's queue, do the task, and then send the results to the master. While the workers are solving the tasks, the master collects the results and puts new tasks on the queue. This is a file-based, remote I/O scheme that serves as the message-passing mechanism between the master and the workers.

The MW paradigm helps the user circumvent the parallel programming challenges, such as load balancing, termination detection, and the distribution of information across compute nodes. Moreover, computation in the MW paradigm is fault-tolerant: if a worker cannot complete a task, due to machine failure or interruption by another user, the master can detect this and put that task back on the queue for another worker to execute. The user can request any number of workers, independent of the number of tasks. HTCondor can make use of a heterogeneous collection of computers, where the fast computers will solve more tasks but slower computers can still contribute.

HTCondor is an example of "High Throughput Computing" (HTC) and is a valuable alternative to "High Performance Computing" (HPC). HPC is typically associated with supercomputers. Its advantage is the specialized communication hardware that allows for rapid communication among pro-

cessors. However, a supercomputer program is assigned a large, but fixed, number of processors; therefore, HPC can be efficient only if an algorithm can keep large numbers of processors busy during the entire computation. Algorithms that need different numbers of processors at different stages cannot be implemented efficiently on HPC architectures. There are also access problems with HPC. Due to the necessity of having a block of processors, users must reserve time, and the lag time between requesting time and getting access increases with the number of desired processors and requested time. Moreover, economists face substantial bureaucratic hurdles in getting access to supercomputer time because the people who control supercomputers impose requirements that are met by few economists. In particular, the authors have been told that DOE supercomputers available to the general scientific community are not available to economists who want to analyze policy issues, such as taxation problems.

In contrast, HTC is a paradigm with much greater flexibility and lower cost. The marginal social cost of CPU time used in HTCondor is essentially zero because it is using CPU time that otherwise would go unused. HTCondor manages the number of processors being used in response to processor availability and the needs of the computational procedure. If HTCondor sees that a computation needs hundreds of processors, it will give the computation what it needs if the resources are available, but if it later sees that a computation needs only a dozen processors, it can free up unused processors and allocate them to other computations. HTC is opportunistic, utilizing any resource that becomes available and not forcing the user to make reservations. The disadvantage of HTC is that interprocessor communication will generally be slower. While this does limit the amount of parallelization that can be exploited, HTC environments can still efficiently use hundreds of processors for many problems. This paper shows that DP is that kind of problem.

For any researcher, the critical measure of computational cost has two components: the time between his submission of a job and when he receives the results, and the time he needs to spend getting access to a computer system. On this dimension, HTC may dominate HPC for any researcher, but even more so for economists where HTC is not just an option but is the only option.

3 Dynamic Programming

In economics and finance, we often encounter a finite horizon optimal decision-making problem that can be expressed in the following general model:

$$V_0(x_0, \theta_0) = \max_{a_t \in \mathcal{D}(x_t, \theta_t, t)} \mathbb{E}\left\{ \sum_{t=0}^{T-1} \beta^t u_t(x_t, a_t) + \beta^T V_T(x_T, \theta_T) \right\},$$

where x_t is a continuous state process with an initial state x_0, θ_t is a discrete state process with an initial state θ_0, and a_t is an action variable (x_t, θ_t and a_t can be vectors), $u_t(x, a)$ is a utility function at time $t < T$ and $V_T(x, \theta)$ is a given terminal value function, β is the discount factor ($0 < \beta \leq 1$), $\mathcal{D}(x_t, \theta_t, t)$ is a feasible set of a_t, and $\mathbb{E}\{\cdot\}$ is the expectation operator.

The DP model for the finite horizon problems is the basic Bellman equation,

$$V_t(x, \theta) = \max_{a \in \mathcal{D}(x, \theta, t)} u_t(x, a) + \beta \mathbb{E}\{V_{t+1}(x^+, \theta^+)\},$$

for $t = 0, 1, \ldots, T-1$, where (x^+, θ^+) is the next-stage state conditional on the current-stage state (x, θ) and action a, and $V_t(x, \theta)$ is called the value function at stage t while the terminal value function $V_T(x, \theta)$ is given.

3.1 Numerical DP Algorithms

In DP problems, if state variables and control variables are continuous, then value functions must be approximated in some computationally tractable manner. It is common to approximate value functions with a finitely parameterized collection of functions; that is, $V(x, \theta) \approx \hat{V}(x, \theta; \mathbf{b})$, where $\mathbf{b}$ is a vector of parameters. The functional form $\hat{V}$ may be a linear combination of polynomials, or it may represent a rational function or neural network representation, or it may be some other parameterization specially designed for the problem. After the functional form is fixed, we focus on finding the vector of parameters, $\mathbf{b}$, such that $\hat{V}(x, \theta; \mathbf{b})$ approximately satisfies the Bellman equation (Bellman, 1957). Algorithm 1 is the parametric DP method with value function iteration for finite horizon problems with both multidimensional continuous and discrete states. (More detailed discussion of numerical DP can be found in Cai (2009), Judd (1998) and Rust (2008).) In the algorithm, n is the dimension for the continuous states x, and d is the dimension for discrete states $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$, where D is

Algorithm 1 Parametric Dynamic Programming with Value Function Iteration for Problems with Multidimensional Continuous and Discrete States

Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$ and the probability transition matrix $P = \left(p_{j,j'}\right)_{D\times D}$ where $p_{j,j'}$ is the transition probability from $\theta^j \in \Theta$ to $\theta^{j'} \in \Theta$ for $1 \leq j, j' \leq D$. Choose a functional form for $\hat{V}(x, \theta; \mathbf{b})$ for all $\theta \in \Theta$, and choose the approximation grid, $\mathbb{X}_t = \{x_t^i : 1 \leq i \leq N_t\} \subset \mathbb{R}^n$. Let $\hat{V}(x, \theta; \mathbf{b}^T) = V_T(x, \theta)$. Then for $t = T-1, T-2, \ldots, 0$, iterate through steps 1 and 2.

Step 1. Maximization step. Compute

$$v_{i,j} = \max_{a \in \mathcal{D}(x^i, \theta^j, t)} u_t(x^i, \theta^j, a) + \beta \mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\},$$

for each $x^i \in \mathbb{X}_t$ and $\theta^j \in \Theta$, $1 \leq i \leq N_t$, $1 \leq j \leq D$, where the next-stage discrete state θ^+ is random with probability mass function $\Pr(\theta^+ = \theta^{j'} \mid \theta^j) = p_{j,j'}$ for each $\theta^{j'} \in \Theta$, and x^+ is the next-stage state transition from x^i and may be also random.

Step 2. Fitting step. Using an appropriate approximation method, for each $1 \leq j \leq D$, compute $\mathbf{b}_j^t$, such that $\hat{V}(x, \theta^j; \mathbf{b}_j^t)$ approximates $\{(x^i, v_{i,j}): 1 \leq i \leq N_t\}$ data, i.e., $v_{i,j} \approx \hat{V}(x^i, \theta^j; \mathbf{b}_j^t)$ for all $x^i \in \mathbb{X}_t$. Let $\mathbf{b}^t = \left\{\mathbf{b}_j^t : 1 \leq j \leq D\right\}$.

---

the number of different discrete state vectors. The transition probabilities from θ^j to $\theta^{j'}$ for $1 \leq j, j' \leq D$ are given.

3.2 Approximation

An approximation scheme has two ingredients: basis functions and approximation nodes. Approximation nodes can be chosen as uniformly spaced nodes, Chebyshev nodes, or some other specified nodes. From the viewpoint of basis functions, approximation methods can be classified as either spectral methods or finite element methods. A spectral method uses globally nonzero basis functions $\phi_j(x)$ such that $\hat{V}(x; \mathbf{b}) = \sum_{j=0}^m b_j \phi_j(x)$. Examples of spectral methods include ordinary polynomial approximation, ordinary Chebyshev polynomial approximation, shape-preserving Chebyshev polynomial approximation (Cai and Judd, 2012b), and Chebyshev-Hermite approximation (Cai and Judd, 2012c). In contrast, a finite element method uses

local basis functions $\phi_j(x)$ that are nonzero over sub-domains of the approximation domain. Examples of finite element methods include piecewise linear interpolation, shape-preserving rational function spline interpolation (Cai and Judd, 2012a), cubic splines, and B-splines. See Cai (2009), Cai and Judd (2010), and Judd (1998) for more details.

3.2.1 Chebyshev Polynomial Approximation

Chebyshev polynomials on $[-1, 1]$ are defined as $\mathcal{T}_j(x) = \cos(j \cos^{-1}(x))$, while general Chebyshev polynomials on $[x_{\min}, x_{\max}]$ are defined as $\mathcal{T}_j((2x - x_{\min} - x_{\max})/(x_{\max} - x_{\min}))$ for $j = 0, 1, 2, \ldots$. These polynomials are orthogonal under the weighted inner product: $\langle f, g \rangle = \int_{x_{\min}}^{x_{\max}} f(x)g(x)w(x)dx$ with the weighting function $w(x) = \left(1 - ((2x - x_{\min} - x_{\max})/(x_{\max} - x_{\min}))^2\right)^{-1/2}$. A degree m Chebyshev polynomial approximation for $V(x)$ on $[x_{\min}, x_{\max}]$ is

$$\hat{V}(x; \mathbf{b}) = \sum_{j=0}^{m} b_j \mathcal{T}_j \left(\frac{2x - x_{\min} - x_{\max}}{x_{\max} - x_{\min}} \right), \tag{1}$$

where $\mathbf{b} = \{b_j\}$ are the Chebyshev coefficients.

If we choose the Chebyshev nodes on $[x_{\min}, x_{\max}]$: $x^i = (z_i + 1)(x_{\max} - x_{\min})/2 + x_{\min}$ with $z_i = -\cos((2i - 1)\pi/(2m'))$ for $i = 1, \ldots, m$', and Lagrange data $\{(x^i, v_i) : i = 1, \ldots, m'\}$ are given (where $v_i = V(x^i)$), then the coefficients b_j in (1) can be easily computed by the Chebyshev regression algorithm (see Judd, 1998).

3.2.2 Multidimensional Complete Chebyshev Approximation

In a d-dimensional approximation problem, let the domain of the value function be

$$\left\{ x = (x_1, \ldots, x_n) : x_j^{\min} \leq x_j \leq x_j^{\max},\ j = 1, \ldots, n \right\},$$

for some real numbers $x_j^{\min}$ and $x_j^{\max}$ with $x_j^{\max} > x_j^{\min}$ for $j = 1, \ldots, n$. Let $x^{\min} = (x_1^{\min}, \ldots, x_n^{\min})$ and $x^{\max} = (x_1^{\max}, \ldots, x_n^{\max})$. Then we denote $[x^{\min}, x^{\max}]$ as the domain. Let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be a vector of nonnegative integers. Let $\mathcal{T}_\alpha(z)$ denote the product $\mathcal{T}_{\alpha_1}(z_1) \cdots \mathcal{T}_{\alpha_n}(z_n)$ for $z = (z_1, \ldots, z_n) \in [-1, 1]^n$. Let

$$Z(x) = \left(\frac{2x_1 - x_1^{\min} - x_1^{\max}}{x_1^{\max} - x_1^{\min}}, \ldots, \frac{2x_n - x_n^{\min} - x_n^{\max}}{x_n^{\max} - x_n^{\min}} \right)$$

for any $x = (x_1, \ldots, x_n) \in [x^{\min}, x^{\max}]$.

Using these notations, the degree-m complete Chebyshev approximation for $V(x)$ is

$$\hat{V}_m(x; \mathbf{b}) = \sum_{0 \le |\alpha| \le m} b_\alpha \mathcal{T}_\alpha \left(Z(x) \right), \tag{2}$$

where $|\alpha| = \sum_{j=1}^n \alpha_j$ for the nonnegative integer vector $\alpha = (\alpha_1, \ldots, \alpha_n)$. So the number of terms with $0 \le |\alpha| = \sum_{j=1}^n \alpha_i \le m$ is $\binom{m+n}{n}$ for the degree-m complete Chebyshev approximation in $\mathbb{R}^n$.

3.3 Numerical Integration

In the objective function of the Bellman equation, we often need to compute the conditional expectation of $V(x^+)$. When the random variable is continuous, we have to use numerical integration to compute the expectation. Gaussian quadrature rules are often applied in computing the integration.

3.3.1 Gauss-Hermite Quadrature

In the expectation operator of the objective function of the Bellman equation, if the random variable has a normal distribution, then it will be good to apply the Gauss-Hermite quadrature formula to compute the numerical integration. That is, if we want to compute $\mathbb{E}\{f(Y)\}$ where Y has a distribution $\mathcal{N}(\mu, \sigma^2)$, then

$$\begin{aligned}
\mathbb{E}\{f(Y)\} &= (2\pi\sigma^2)^{-1/2} \int_{-\infty}^{\infty} f(y) e^{-(y-\mu)^2/(2\sigma^2)} dy \\
&= (2\pi\sigma^2)^{-1/2} \int_{-\infty}^{\infty} f(\sqrt{2}\,\sigma\, x + \mu) e^{-x^2} \sqrt{2}\sigma dx \\
&\doteq \pi^{-\frac{1}{2}} \sum_{i=1}^{m} \omega_i f(\sqrt{2}\sigma x_i + \mu),
\end{aligned}$$

where ω_i and x_i are the Gauss-Hermite quadrature with m weights and nodes over $(-\infty, \infty)$. See Cai (2009), Judd (1998), Stroud and Secrest (1966) for more details.

If Y is log normal, i.e., $\log(Y)$ has a distribution $\mathcal{N}(\mu, \sigma^2)$, then we can

assume that $Y = e^X$ where $X \sim \mathcal{N}(\mu, \sigma^2)$, thus

$$\mathbb{E}\{f(Y)\} = \mathbb{E}\{f(e^X)\} \doteq \pi^{-\frac{1}{2}} \sum_{i=1}^{m} \omega_i f\left(e^{\sqrt{2}\sigma x_i + \mu}\right).$$

3.3.2 Multidimensional Integration

If we want to compute a multidimensional integration, we could apply the product rule. For example, suppose that we want to compute $\mathbb{E}\{f(X)\}$, where X is a random vector with multivariate normal distribution $\mathcal{N}(\mu, \Sigma)$ over $\mathbb{R}^n$, where μ is the mean column vector and Σ is the covariance matrix, then we could do the Cholesky factorization first, i.e., find a lower triangular matrix L such that $\Sigma = LL^\top$. This is feasible as Σ must be a positive semi-definite matrix from the covariance property. Thus,

$$\begin{aligned}
\mathbb{E}\{f(X)\} &= ((2\pi)^n \det(\Sigma))^{-1/2} \int_{\mathbb{R}^n} f(y) e^{-(y-\mu)^\top \Sigma^{-1}(y-\mu)/2} dy \\
&= \left((2\pi)^n \det(L)^2\right)^{-1/2} \int_{\mathbb{R}^n} f\left(\sqrt{2}Lx + \mu\right) e^{-x^\top x} 2^{n/2} \det(L) dx \\
&\doteq \pi^{-\frac{n}{2}} \sum_{i_1=1}^{m} \cdots \sum_{i_n=1}^{m} \omega_{i_1} \cdots \omega_{i_d} f\Bigg(\sqrt{2} l_{1,1} x_{i_1} + \mu_1, \\
&\qquad \sqrt{2}(l_{2,1} x_{i_1} + l_{2,2} x_{i_2}) + \mu_2, \cdots, \sqrt{2}\left(\sum_{j=1}^{n} l_{n,j} x_{i_j}\right) + \mu_n\Bigg), \quad (3)
\end{aligned}$$

where ω_i and x_i are the Gauss-Hermite quadrature with m weights and nodes over $(-\infty, \infty)$, $l_{i,j}$ is the (i,j)-element of L, and $\det(\cdot)$ means the matrix determinant operator.

4 Parallel Dynamic Programming

The numerical DP algorithms can be applied easily in the HTCondor MW system for DP problems with multidimensional continuous and discrete states. To solve these problems, numerical DP algorithms with value function iteration have the maximization step that is mostly time-consuming in numerical DP. That is,

$$v_{i,j} = \max_{a \in \mathcal{D}(x^i, \theta^j, t)} u(x^i, \theta^j, a) + \beta \mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\},$$

Algorithm 2 Type-I Parallel Dynamic Programming with Value Function Iteration for the Master

Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$. Set $\mathbf{b}^T$ as the parameters of the terminal value function. For $t = T-1, T-2, \ldots, 0$, iterate through steps 1 and 2.

Step 1. Separate the maximization step into D tasks, one task per $\theta \in \Theta$. Each task contains parameters $\mathbf{b}^{t+1}$, stage number t and the corresponding task identity for some θ^j. Then send these tasks to the workers.

Step 2. Wait until all tasks are done by the workers. Then collect parameters $\mathbf{b}_j^t$ from the workers, for all $1 \leq j \leq D$, and let $\mathbf{b}^t = \left\{\mathbf{b}_j^t : 1 \leq j \leq D\right\}$.

for each continuous state point x^i in the finite set $\mathbb{X}_t \subset \mathbb{R}^n$ and each discrete state vector $\theta^j \in \Theta$, where N_t is the number of points of $\mathbb{X}_t$ and D is the number of points of Θ. So there are $N_t \times D$ small-size maximization problems. Thus, if the $N_t \times D$ is large (that is very possible in high-dimensional problems), then it will take a huge amount of time to do the DP maximization step. However, these $N_t \times D$ small-size maximization problems can be naturally parallelized in the HTCondor MW system, in which one or several maximization problem(s) could be treated as one task.

4.1 Type-I Parallelization

When D is large but N_t has a medium size, we could separate the $N_t \times D$ maximization problems into D tasks, where each task corresponds to a discrete state vector θ^j and all continuous state nodes set $\mathbb{X}_t$. Algorithm 2 is the architecture for the master processor, and Algorithm 3 is the corresponding architecture for the workers.

4.2 Type-II Parallelization

If the number of nodes for continuous states, N_t, is large, or the maximization step for each node is time-consuming, then it will be possible to break the task for one θ^j into subtasks and maintain parallel efficiency. If the fitting method requires all points $\{(x^i, v_{i,j}) \colon 1 \leq i \leq N_t\}$ to construct the

Algorithm 3 Type-I Parallel Dynamic Programming with Value Function Iteration for the Workers

Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \le j \le D\} \subset \mathbb{R}^d$ and the probability transition matrix $P = \left(p_{j,j'}\right)_{D\times D}$ where $p_{j,j'}$ is the transition probability from $\theta^j \in \Theta$ to $\theta^{j'} \in \Theta$ for $1 \le j, j' \le D$. Choose a functional form for $\hat{V}(x, \theta; \mathbf{b})$ for all $\theta \in \Theta$.

Step 1. Get parameters $\mathbf{b}^{t+1}$, stage number t and the corresponding task identity for one $\theta^j \in \Theta$ from the master, and then choose the approximation grid, $\mathbb{X}_t = \{x_t^i : 1 \le i \le N_t\} \subset \mathbb{R}^n$.

Step 2. For this given θ^j, compute

$$v_{i,j} = \max_{a\in\mathcal{D}(x^i,\theta^j,t)} u(x^i, \theta^j, a) + \beta\mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\},$$

for each $x^i \in \mathbb{X}_t$, $1 \le i \le N_t$, where the next-stage discrete state $\theta^+ \in \Theta$ is random with probability mass function $\mathbb{P}(\theta^+ = \theta^{j'} \mid \theta^j) = p_{j,j'}$ for each $\theta_{j'} \in \Theta$, and x^+ is the next-stage state transition from x^i and may be also random.

Step 3. Using an appropriate approximation method, compute $\mathbf{b}_j^t$ such that $\hat{V}(x, \theta^j; \mathbf{b}_j^t)$ approximates $\{(x^i, v_{i,j})\colon\ 1 \le i \le N_t\}$, i.e., $v_{i,j} \approx \hat{V}(x^i, \theta^j; \mathbf{b}_j^t)$ for all $x^i \in \mathbb{X}_t$.

Step 4. Send $\mathbf{b}_j^t$ and the corresponding task identity for θ^j to the master.

Algorithm 4 Type-II Parallel Dynamic Programming with Value Function Iteration for the Master

Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$. Choose a functional form for $\hat{V}(x, \theta; \mathbf{b})$ for all $\theta \in \Theta$, and choose the approximation grid, $\mathbb{X}_t = \{x_t^i : 1 \leq i \leq N_t\} \subset \mathbb{R}^n$. Set $\mathbf{b}^T$ as the parameters of the terminal value function. For $t = T-1, T-2, \ldots, 0$, iterate through steps 1 and 2.

Step 1. Separate $\mathbb{X}_t$ into M disjoint subsets with almost equal sizes: $\mathbb{X}_{t,1}, \ldots, \mathbb{X}_{t,M}$, and separate the maximization step into $M \times D$ tasks, one task per $(\mathbb{X}_{t,m}, \theta^j)$ with $\theta^j \in \Theta$, for $m = 1, \ldots, M$ and $j = 1, \ldots, D$. Each task contains the parameters $\mathbf{b}^{t+1}$, the stage number t and the corresponding task identity for $(\mathbb{X}_{t,m}, \theta^j)$. Then send these tasks to the workers.

Step 2. Wait until all tasks are done by the workers. Then collect all $v_{i,j}$ from the workers, for $1 \leq i \leq N_t$, $1 \leq j \leq D$.

Step 3. Using an appropriate approximation method, for each $\theta^j \in \Theta$, compute $\mathbf{b}_j^t$ such that $\hat{V}(x, \theta^j; \mathbf{b}_j^t)$ approximates $\{(x^i, v_{i,j})\colon\ 1 \leq i \leq N_t\}$, i.e., $v_{i,j} \approx \hat{V}(x^i, \theta^j; \mathbf{b}_j^t)$ for all $x^i \in \mathbb{X}_t$. Let $\mathbf{b}^t = \left\{\mathbf{b}_j^t : 1 \leq j \leq D\right\}$.

approximation, then each worker cannot do step 3 and 4 along with step 1 and 2 in Algorithm 3, as it has only an incomplete set of approximation nodes x^i for one given θ^j. Therefore, the fitting step is executed by the master. Thus we have Algorithm 4 for the master process and Algorithm 5 for the workers.

If it is quick to compute $\mathbf{b}_j^t$ in the fitting step (e.g., Chebyshev polynomial approximation using Chebyshev regression algorithm), then we can just let the master do the fitting step like the type-II parallel DP algorithm. However, if the fitting step is time-consuming, then the master could send these fitting jobs for each discrete state θ^j to the workers, and then collect the the new approximation parameters.

4.3 Sparsity

In many cases, the probability transition matrix is sparse and this fact can be exploited to reduce communication cost. For example, suppose that a worker is given the task to compute the value function for θ^j. When

Algorithm 5 Type-II Parallel Dynamic Programming with Value Function Iteration for the Workers

Initialization. Given a finite set of $\theta \in \Theta = \{\theta^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$ and the probability transition matrix $P = \left(p_{j,j'}\right)_{D\times D}$ where $p_{j,j'}$ is the transition probability from $\theta^j \in \Theta$ to $\theta^{j'} \in \Theta$ for $1 \leq j, j' \leq D$. Choose the approximation grid, $\mathbb{X}_t = \{x_t^i : 1 \leq i \leq N_t\} \subset \mathbb{R}^n$, which is the same with the set $\mathbb{X}_t$ in the master.

Step 1. Get the parameters $\mathbf{b}^{t+1}$, stage number t and the corresponding task identity for one $(\mathbb{X}_{t,m}, \theta^j)$ with $\theta^j \in \Theta$ from the master.

Step 2. For this given θ^j, compute

$$v_{i,j} = \max_{a\in\mathcal{D}(x^i,\theta^j,t)} u(x^i, \theta^j, a) + \beta\mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\},$$

for all $x^i \in \mathbb{X}_{t,m}$, where the next-stage discrete state $\theta^+ \in \Theta$ is random with probability mass function $\mathbb{P}(\theta^+ = \theta^{j'} \mid \theta^j) = p_{j,j'}$ for each $\theta^{j'} \in \Theta$, and x^+ is the next-stage state transition from x^i and may be also random.

Step 3. Send $v_{i,j}$ for these given $x^i \in \mathbb{X}_{t,m}$ and θ^j, to the master process.

---

it computes the expectation in the objective function of the maximization problems, it only needs access to the value functions for those $\theta^{j'}$ which can reached from θ^j in one period. That is,

$$\mathbb{E}\{\hat{V}(x^+, \theta^+; \mathbf{b}^{t+1})\} = \sum_{1\leq j'\leq D,\, p_{j,j'}\neq 0} p_{j,j'}\mathbb{E}\{\hat{V}(x^+, \theta^{j'}; \mathbf{b}_{j'}^{t+1})\}.$$

Therefore, when the master forms the description of a task for a worker, it only needs to include those $\mathbf{b}_{j'}^{t+1}$ with nonzero transition probability $p_{j,j'}$ (instead of the whole set of parameters, $\mathbf{b}^{t+1}$) in the tasks corresponding to θ^j, i.e., $\left\{\mathbf{b}_{j'}^{t+1} : p_{j,j'} > 0,\, 1 \leq j' \leq D\right\}$ where $p_{j,j'} = \mathbb{P}(\theta^+ = \theta^{j'} \mid \theta^j)$, and then send this subset of $\mathbf{b}^{t+1}$ to the workers in Step 1 of Algorithm 2 or 4. This saves on master-worker communication costs.

5 Application on Stochastic Optimal Growth Models

We consider a multi-dimensional stochastic optimal growth problem. We assume that there are d sectors, and let $k_t = (k_{t,1}, \ldots, k_{t,d})$ denote the capital stocks of these sectors which is a d-dimensional continuous state vector at time t. Let $\theta_t = (\theta_{t,1}, \ldots, \theta_{t,d}) \in \Theta = \{\theta_t^j : 1 \leq j \leq D\} \subset \mathbb{R}^d$ denote current productivity levels of the sectors which is a d-dimensional discrete state vector at time t, and assume that θ_t follows a Markov process with a stable probability transition matrix, denoted as $\theta_{t+1} = g(\theta_t, \xi_t)$ where ξ_t are i.i.d. disturbances. Let $l_t = (l_{t,1}, \ldots, l_{t,d})$ denote elastic labor supply levels of the sectors which is a d-dimensional continuous control vector variable at time t. Assume that the net production function of sector i at time t is $f(k_{t,i}, l_{t,i}, \theta_{t,i})$, for $i = 1, \ldots, d$. Let $c_t = (c_{t,1}, \ldots, c_{t,d})$ and $I_t = (I_{t,1}, \ldots, I_{t,d})$ denote, respectively, consumption and investment of the sectors at time t. We want to find an optimal consumption and labor supply decisions such that expected total utility over a finite-horizon time is maximized, i.e.,

$$
\begin{aligned}
V_0(k_0, \theta_0) \quad = \quad & \max_{k_t, I_t, c_t, l_t} \quad \mathbb{E}\left\{ \sum_{t=0}^{T-1} \beta^t u(c_t, l_t) + \beta^T V_T(k_T, \theta_T) \right\}, \\
& \text{s.t.} \quad k_{t+1,j} = (1-\delta) k_{t,j} + I_{t,j} + \epsilon_{t,j}, \quad j = 1, \ldots, d, \\
& \qquad \Gamma_{t,j} = \frac{\zeta}{2} k_{t,j} \left(\frac{I_{t,j}}{k_{t,j}} - \delta \right)^2, \quad j = 1, \ldots, d, \\
& \qquad \sum_{j=1}^{d} \left(c_{t,j} + I_{t,j} - \delta k_{t,j} \right) = \sum_{j=1}^{d} \left(f(k_{t,j}, l_{t,j}, \theta_{t,j}) - \Gamma_{t,j} \right), \\
& \qquad \theta_{t+1} = g(\theta_t, \xi_t),
\end{aligned}
$$

where k_0 and θ_0 are given, δ is the depreciation rate of capital, $\Gamma_{t,j}$ is the investment adjustment cost of sector j, and ζ governs the intensity of the friction, $\epsilon_t = (\epsilon_{t,1}, \ldots, \epsilon_{t,d})$ are serially uncorrellated i.i.d. disturbances with $\mathbb{E}\{\epsilon_{t,i}\} = 0$, and $V_T(k, \theta)$ is a given terminal value function. For this finite-horizon model, Cai and Judd (2012c) solve some of its simplified problem. An infinite-horizon version of this model is introduced in Den Haan et al (2011), Juillard and Villemot (2011), and a nonlinear programming method for dynamic programming is introduced in Cai et al. (2013a) to solve the

multi-country growth model with infinite horizon.

5.1 Dynamic Programming Model

The DP formulation of the multi-dimensional stochastic optimal growth problem is

$$\begin{aligned} V_t(k,\theta) \;=\; \max_{c,l,I} \;& u(c,l) + \beta \mathbb{E}\left\{ V_{t+1}(k^+,\theta^+) \mid \theta \right\}, \\ \text{s.t.} \;& k_j^+ = (1-\delta)k_j + I_j + \epsilon_j, \quad j = 1,\ldots,d, \\ & \Gamma_j = \frac{\zeta}{2} k_j \left(\frac{I_j}{k_j} - \delta \right)^2, \quad j = 1,\ldots,d, \\ & \sum_{j=1}^{d} (c_j + I_j - \delta k_j) = \sum_{j=1}^{d} \left(f(k_j, l_j, \theta_j) - \Gamma_j \right), \\ & \theta^+ = g(\theta, \xi_t), \end{aligned}$$

for $t = 0,\ldots,T-1$, where $k = (k_1,\ldots,k_d)$ is the continuous state vector and $\theta = (\theta_1,\ldots,\theta_d) \in \Theta = \{(\vartheta_{j,1},\ldots,\vartheta_{j,d}) : 1 \le j \le D\}$ is the discrete state vector, $c = (c_1,\ldots,c_d)$, $l = (l_1,\ldots,l_d)$, and $I = (I_1,\ldots,I_d)$ are control variables, $\epsilon = (\epsilon_1,\ldots,\epsilon_d)$ are i.i.d. disturbance with mean 0, and $k^+ = (k_1^+,\ldots,k_d^+)$ and $\theta^+ = \left(\theta_1^+,\ldots,\theta_d^+\right) \in \Theta$ are the next-stage state vectors. Numerically, $V(k,\theta)$ is approximated with given values at finite nodes, so the approximation is only good at a finite range. That is, the state variable must be in a finite range $[\underline{k},\bar{k}]$, then we should have the restriction $k^+ \in [\underline{k},\bar{k}]$. Here $\underline{k} = (\underline{k}_1,...,\underline{k}_d), \bar{k} = (\bar{k}_1,...,\bar{k}_d)$, and $k^+ \in [\underline{k},\bar{k}]$ denotes that $k_i^+ \in [\underline{k}_i,\bar{k}_i]$ for all $1 \le i \le d$. Moreover, we should add $c > 0$ and $l > 0$ in the constraints.

5.2 Numerical Example

In the following numerical example, we see the application of parallelization of numerical DP algorithms for the DP model of the multi-dimensional stochastic optimal growth problem. We let $T = 5$, $\beta = 0.8$, $\delta = 0.025$, $\zeta = 0.5$, $[\underline{k},\bar{k}] = [0.2, 3.0]^d$, $f(k_i,l_i,\theta_i) = \theta_i A k_i^{\psi} l_i^{1-\psi}$ with $\psi = 0.36$ and $A = (1-\beta)/(\psi\beta) = 1$, for $i = 1,\ldots,d$, and

$$u(c,l) = \sum_{i=1}^{d} \left[\frac{(c_i/A)^{1-\gamma} - 1}{1-\gamma} - (1-\psi) \frac{l_i^{1+\eta} - 1}{1+\eta} \right],$$

with $\gamma = 2$ and $\eta = 1$.

In this example, we let $d = 4$. So this is a DP example with 4-dimensional continuous states and 4-dimensional discrete states. Here we assume that the possible values of θ_i and θ_i^+ are

$$\vartheta_1 = 0.85,\ \vartheta_2 = 0.9,\ \vartheta_3 = 0.95,\ \vartheta_4 = 1.0,\ \vartheta_5 = 1.05,\ \vartheta_6 = 1.1,\ \vartheta_7 = 1.15,$$

and the probability transition matrix from θ_i to θ_i^+ is a 7×7 tridiagonal matrix:

$$P = \begin{bmatrix} 0.75 & 0.25 & & & & & \\ 0.25 & 0.50 & 0.25 & & & & \\ & 0.25 & 0.50 & 0.25 & & & \\ & & 0.25 & \ddots & \ddots & & \\ & & & \ddots & 0.50 & 0.25 & \\ & & & & 0.25 & 0.75 & \end{bmatrix},$$

for each $i = 1, \ldots, 4$, and we assume that $\theta_1^+, \ldots, \theta_d^+$ are independent of each other. That is,

$$\Pr[\theta^+ = (\vartheta_{i_1}, \ldots, \vartheta_{i_4}) \mid \theta = (\vartheta_{j_1}, \ldots, \vartheta_{j_4})] = P_{i_1,j_1} P_{i_2,j_2} P_{i_3,j_3} P_{i_4,j_4},$$

where P_{i_α,j_α} is the (i_α, j_α) element of P, for any $i_\alpha, j_\alpha = 1, \ldots, 7$, $\alpha = 1, \ldots, 4$.

In addition, we assume that $\epsilon_1, \ldots, \epsilon_4$ are i.i.d., and each ϵ_i has 3 discrete values:

$$\delta_1 = -0.01,\ \delta_2 = 0.0,\ \delta_3 = 0.01,$$

while their probabilities are $q_1 = 0.25$, $q_2 = 0.5$ and $q_3 = 0.25$, respectively. That is,

$$\Pr[\epsilon = (\delta_{n_1}, \ldots, \delta_{n_4})] = q_{n_1} q_{n_2} q_{n_3} q_{n_4},$$

for any $n_\alpha = 1, 2, 3$, $\alpha = 1, \ldots, 4$. Moreover, $\epsilon_1, \ldots, \epsilon_4$ are assumed to be independent of $\theta_1^+, \ldots, \theta_4^+$.

Therefore,

$$\begin{aligned} & \mathbb{E}\{V(k^+, \theta^+) \mid \theta = (\vartheta_{j_1}, \ldots, \vartheta_{j_4})\} \\ = & \sum_{n_1,n_2,n_3,n_4=1}^{3} q_{n_1} q_{n_2} q_{n_3} q_{n_4} \sum_{i_1,i_2,i_3,i_4=1}^{6} P_{i_1,j_1} P_{i_2,j_2} P_{i_3,j_3} P_{i_4,j_4} \times \\ & V(\hat{k}_1^+ + \delta_{n_1}, \ldots, \hat{k}_4^+ + \delta_{n_4}, \vartheta_{i_1}, \ldots, \vartheta_{i_4}), \end{aligned} \tag{4}$$

where $\hat{k}_\alpha^+ = (1-\delta)k_\alpha + I_\alpha$, for any $i_\alpha = 1, \ldots, 7$, $\alpha = 1, \ldots, 4$.

From the formula (4), it seems that we should compute the value function V at a large number of points up to $3^4 * 7^4 = 194,481$ in order to evaluate the expectation. But in fact, we can take advantage of the sparsity of the probability transition matrix P. After canceling the zero probability terms, the evaluation of the expectation will need to compute the value function at a number of points ranging from $3^4 * 2^4 = 1,296$ to $3^4 * 3^4 = 6,561$, which is far less that the case without using the sparsity. Moreover, the communication cost between the master and workers is also far less than the case without using the sparsity.

The continuous value function approximation is the complete degree-6 Chebyshev polynomial approximation method (2) with $7^4 = 2401$ Chebyshev nodes for continuous state variables, the optimizer is NPSOL (Gill, P., et al., 1994), and the terminal value function is chosen as

$$V_T(k, \theta) = u(f(k, \mathbf{e}, \mathbf{e}), \mathbf{e})/(1-\beta),$$

where $\mathbf{e}$ is the vector with 1's everywhere. Here $\mathbf{e}$ is chosen because it is the steady state labor supply for the corresponding infinite-horizon problem and is also the average value of θ.

5.3 HTCondor-MW Results

We use the master algorithm 2 and the worker algorithm 3 to solve the optimal growth problem. There are seven possible values of θ_i for each $i = 1, \ldots, 4$, and each task consists of updating the value function at one specific θ^j; therefore, the total number of HTCondor-MW tasks for one value function iteration is $7^4 = 2401$. Furthermore, we use seven approximation nodes in each continuous dimension to construct a degree six complete polynomial; therefore, each task computes 2401 small-size maximization prob-

Table 1: Statistics of Parallel DP under HTCondor-MW for the growth problem

Wall clock time for all 3 VFIs	8.28 hours
Wall clock time for 1st VFI	0.34 hours
Wall clock time for 2nd VFI	3.92 hours
Wall clock time for 3rd VFI	4.01 hours
Total time workers were up (alive)	16.9 days
Total cpu time used by all workers	16.5 days
Number of (different) workers	50
Average Number Present Workers	49
Overall Parallel Performance	98.6%

lems as there are 2401 Chebyshev nodes.

Under HTCondor, we assign 50 workers to do this parallel work. Table 1 lists some statistics of our parallel DP algorithm under HTCondor-MW system for the growth problem after running 3 value function iterations (VFI). The last line of Table 1 shows that the parallel efficiency of our parallel numerical DP method is very high (up to 98.6%) for this example. We see that the total cpu time used by all workers to solve the optimal growth problem is nearly 17 days, i.e., it will take nearly 17 wall clock days to solve the problem without using parallelism. However, it takes only 8.28 wall clock hours to solve the problem if we use the parallel algorithm and 50 worker processors.

Table 2 gives the parallel efficiency with various number of worker processors for this optimal growth model. We see that it has an almost linear speed-up when we add the number of worker processors from 50 to 200. We see that the wall clock time to solve the problem is only 2.26 hours now if the number of worker processors increases to 200.

Parallel efficiency drops from 99% to 92% when we move from 100 processors to 200. This is not the critical fact for a user. The most important fact is that requesting 200 processors reduced the waiting time from submission to final output by 1.6 hours. Focussing on the user's waiting time is one of the values of the HTC approach to parallelization.

Table 2: Parallel efficiency for various numbers of worker processors

# Worker processors	Parallel efficiency	Average task wall clock time (second)	Total wall clock time (hour)
50	98.6%	199	8.28
100	97%	185	3.89
200	91.8%	186	2.26

6 Application to Dynamic Portfolio Problems with Transaction Costs

We consider a dynamic portfolio problem with transaction costs. We assume that an investor begins with some initial wealth W_0, invests it in several assets, and manages it at every time t so as to maximize the expected utility of wealth at a terminal time T. We assume a power utility function for terminal wealth, $u(W) = W^{1-\gamma}/(1-\gamma)$ where $\gamma > 0$ and $\gamma \neq 1$. Let $R = (R_1, \ldots, R_n)^\top$ be the random one-period return of n risky assets, and R_f be the return of the riskless asset. The portfolio share for asset i at the beginning of period t is denoted $x_{t,i}$, and let $x_t = (x_{t,1}, \ldots, x_{t,n})^\top$. The difference between wealth and the wealth invested in stocks is invested in bonds. At the beginning of every period, the investor has a chance to rebalance the portfolio with a proportional transaction cost rate τ for buying or selling stocks. Let $\delta^+_{t,i}W$ denote the amount of asset i purchased, expressed as a fraction of wealth, and let $\delta^-_{t,i}W$ denote the amount sold, where $\delta^+_{t,i}, \delta^-_{t,i} \geq 0$, for periods $t = 0, \ldots, T-1$.

We assume that the riskless return R_f and the risky assets' return R may be dependent on a discrete time stochastic process θ_t (could be a vector), denoted by $R_f(\theta_t)$ and $R(\theta_t)$ respectively, for $t = 0, \ldots, T-1$. Then the

dynamic portfolio problem becomes

$$
\begin{aligned}
V_0(W_0, x_0, \theta_0) \;=\; & \max_{\delta^+, \delta^- \geq 0} \; \mathbb{E}\left\{u(W_T)\right\}, \qquad (5)\\
& \text{s.t.} \quad W_{t+1} = \mathbf{e}^\top X_{t+1} + R_f(\theta_t)(1 - \mathbf{e}^\top x_t - y_t)W_t,\\
& \qquad X_{t+1,i} = R_i(\theta_t)(x_{t,i} + \delta^+_{t,i} - \delta^-_{t,i})W_t,\\
& \qquad y_t = \mathbf{e}^\top(\delta^+_t - \delta^-_t + \tau(\delta^+_t + \delta^-_t)),\\
& \qquad x_{t+1,i} = X_{t+1,i}/W_{t+1},\\
& \qquad \theta_{t+1} = g(\theta_t, \xi_t),\\
& \qquad t = 0, \ldots, T-1; \quad i = 1, \ldots, n,
\end{aligned}
$$

where $\mathbf{e}$ is the column vector with 1's everywhere, $X_{t+1} = (X_{t+1,1}, \ldots, X_{t+1,n})^\top$, $\delta^+_t = (\delta^+_{t,1}, \ldots, \delta^+_{t,n})^\top$, and $\delta^-_t = (\delta^-_{t,1}, \ldots, \delta^-_{t,n})^\top$. Here, W_{t+1} is time $t+1$ wealth, $X_{t+1,i}$ is time $t+1$ wealth in asset i, $y_t W_t$ is the change in bond holding, and $x_{t+1,i}$ is the allocation of risky asset i.

6.1 Dynamic Programming Model

The DP model of the multi-stage portfolio optimization problem (5) is

$$
V_t(W, x, \theta) \;=\; \max_{\delta^+, \delta^- \geq 0} \; \mathbb{E}\left\{V_{t+1}(W^+, x^+, \theta^+)\right\},
$$

for $t = 0, 1, \ldots, T-1$, while the terminal value function is $V_T(W, x, \theta) = W^{1-\gamma}/(1-\gamma)$. Given the isoelasticity of V_T, we know that the value function can be rewritten as

$$
V_t(W_t, x_t, \theta_t) = W_t^{1-\gamma} \cdot H_t(x_t, \theta_t),
$$

for some functions $H_t(x_t, \theta_t)$, where W_t and x_t are respectively wealth and allocation fractions of stocks right before re-balancing at stage $t = 0, 1, \ldots, T$,

and

$$
\begin{aligned}
H_t(x,\theta) \;=\; & \max_{\delta^+,\delta^-} \; \mathbb{E}\left\{\Pi^{1-\gamma}\cdot H_{t+1}(x^+,\theta^+)\right\}, \qquad (6)\\
& \text{s.t.} \quad \delta^+ \geq 0,\ \delta^- \geq 0,\\
& \quad x+\delta^+-\delta^- \geq 0,\\
& \quad y \leq 1-\mathbf{e}^\top x,\\
& \quad \theta^+ = g(\theta,\xi_t),
\end{aligned}
$$

where $H_T(x,\theta) = 1/(1-\gamma)$, and

$$
\begin{aligned}
y &\equiv \mathbf{e}^\top(\delta^+-\delta^-+\tau(\delta^++\delta^-)),\\
s_i &\equiv R_i(\theta)(x_i+\delta_i^+-\delta_i^-),\\
\Pi &\equiv \mathbf{e}^\top s + R_f(\theta)(1-\mathbf{e}^\top x-y),\\
x_i^+ &\equiv s_i/\Pi,
\end{aligned}
$$

for $i = 1,\ldots,n$ and $t = 0,1,\ldots,T-1$. See Cai, Judd and Xu (2013b) for a detailed discussion of this dynamic portfolio optimization problem.

Since W_t and x_t are separable, we can just assume that $W_t = 1$ dollar for simplicity. Thus, at time t, δ^+ and δ^- are the amounts for buying and selling stocks respectively, y is the change in bond holding, s is the next-stage amount vector of dollars on the stocks, Π is the total wealth at the next stage, and x^+ is the new fraction vector of the stocks at the next stage. In this model, the state variables, x and x^+, are continuous in $[0,1]^n$.

6.2 Numerical Examples

We choose a portfolio with $n = 6$ stocks and one riskless bond. The investor wants to maximize the expected terminal utility after $T = 6$ years with the terminal utility, $u(W) = W^{1-\gamma}/(1-\gamma)$, with $\gamma = 4$. At the beginning of each year $t = 0,1,\ldots T-1$, the investor has a chance to rebalance the portfolio with a proportional transaction cost rate $\tau = 0.002$ for buying or selling stocks. We assume that the stock returns are independent each other, and stock i has a log-normal annual return, i.e., $\log(R_i) \sim \mathcal{N}(\mu_i - \sigma_i^2/2,\, \sigma_i^2)$ with $\mu_i = 0.07$ and $\sigma_i = 0.25$, for $i = 1,\ldots,n$. We assume that the bond has a riskless annual return $\exp(r_t)$, while the interest rate r_t is a discrete Markov chain, with $r_t = 0.01, 0.02, 0.03, 0.04$ or 0.05, and its transition probability

matrix is

$$P = \begin{bmatrix} 0.7 & 0.3 & & & \\ 0.3 & 0.4 & 0.3 & & \\ & 0.3 & 0.4 & 0.3 & \\ & & 0.3 & 0.4 & 0.3 \\ & & & 0.3 & 0.7 \end{bmatrix}.$$

We use the degree-4 complete Chebyshev polynomials (2) as the approximation method, and choose 5 Chebyshev nodes on each dimension, so that we can apply the Chebyshev regression algorithm to compute the approximation coefficients in the fitting step of numerical DP algorithms. Thus, the number of approximation nodes is $5^6 = 15,625$ for each discrete state, so the total number of small-size maximization problems for one value function iteration is $5 \times 5^6 = 78,125$. We use the product Gauss-Hermite quadrature formula (3) with 5 nodes for each dimension, so the number of quadrature nodes is $5^6 = 15,625$ for each discrete state. Therefore, after using the sparsity of the probability transition matrix, the computation of the expectation in the objective function of the maximization problem (6) includes $2 \times 5^6 = 31,250$ or $3 \times 5^6 = 46,875$ evaluations of the approximated value function at stage $t+1$ for each approximation node. We use NPSOL as our optimization solver for solving the maximization problem (6) .

6.3 HTCondor-MW Results

We apply Algorithm 4 and 5 to solve the high-dimensional dynamic portfolio problem. Each HTCondor-MW task solves 25 small-size maximization problems, implying that each value function iteration is broken into 3,125 MTCondor-MW tasks. Our HTCondor program requested 200 workers, and was given 194 processors on average.

Table 3 lists some statistics of our parallel DP algorithm under HTCondor-MW system for the portfolio problem with six stocks and one bond with stochastic interest rates. The parallel efficiency of our parallel numerical DP method is 94.2% for this example, even when we use 200 workers. Moreover, the total cpu time used by all workers to solve the dynamic portfolio optimization problem is more than 27 days, i.e., it will take more than 27 days to solve the problem using a single core. However, it takes only about 3.6 wall clock hours to solve the problem if we use the type-II parallel DP algorithm and 200 worker processors. This reduction in "waiting time" cost

Table 3: Statistics of Parallel DP under HTCondor-MW for the 7-asset portfolio problem with stochastic interest rate

Wall clock time for all 6 VFIs	3.6 hours
Wall clock time for 1st VFI	4.8 minutes
Wall clock time for 2nd VFI	43.4 minutes
Wall clock time for 3rd VFI	40.6 minutes
Wall clock time for 4th VFI	41.5 minutes
Wall clock time for 5th VFI	42.9 minutes
Wall clock time for 6th VFI	43.7 minutes
Total time workers were up (alive)	29.3 days
Total cpu time used by all workers	27.4 days
Number of (different) workers	200
Average Number Present Workers	194
Overall Parallel Performance	94.2%

to a researcher makes it possible to solve problems that essentially cannot be solved on a laptop.

7 Conclusion

This paper presents the parallel dynamic programming methods in HTCondor Master-Worker system. That system can be used to solve very demanding high-dimensional dynamic programming problems efficiently. While we only used DP examples, the simple structure of parallelization used for DP problems is similar to parallelization strategies that can be used for many other economic problems, such as computing high-dimensional dynamic general equilibrium problems. HTCondor Master-Worker is clearly a powerful tool with many potential applications for economists.

References

[1] Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

[2] Cai, Y. (2009). *Dynamic Programming and Its Application in Economics and Finance*. PhD thesis, Stanford University.

[3] Cai, Y., and K.L. Judd (2010). Stable and efficient computational methods for dynamic programming. *Journal of the European Economic Association*, Vol. 8, No. 2-3, 626–634.

[4] Cai, Y., and K.L. Judd (2012a). Dynamic programming with shape-preserving rational spline Hermite interpolation. *Economics Letters*, Vol. 117, No. 1, 161–164.

[5] Cai, Y., and K.L. Judd (2012b). Shape-preserving dynamic programming. *Mathematical Methods of Operations Research*, DOI: 10.1007/s00186-012-0406-5.

[6] Cai, Y., and K.L. Judd (2012c). Dynamic programming with Hermite approximation. NBER working paper No. w18540.

[7] Cai, Y., K.L. Judd, T.S. Lontzek, V. Michelangeli, and C.-L. Su (2013a). Nonlinear programming method for dynamic programming. Hoover working paper.

[8] Cai, Y., K.L. Judd and R. Xu (2013b). Numerical solutions of dynamic portfolio optimization with transaction costs. Hoover working paper.

[9] Den Haan, W.J., K.L. Judd and M. Juillard (2011). Computational suite of models with heterogeneous agents II: Multi-country real business cycle models. Journal of Economic Dynamics & Control, 35, 175–177.

[10] Judd, K.L. (1998). *Numerical Methods in Economics*. The MIT Press.

[11] Gill, P., W. Murray, M.A. Saunders and M.H. Wright (1994). User's Guide for NPSOL 5.0: a Fortran Package for Nonlinear Programming. Technical report, SOL, Stanford University.

[12] Juillard, M., and S. Villemot (2011). Multi-country real business cycle models: Accuracy tests and test bench. *Journal of Economic Dynamics & Control*, 35, 178–185.

[13] Rust, J. (2008). Dynamic Programming. In: *New Palgrave Dictionary of Economics*, ed. by Steven N. Durlauf and Lawrence E. Blume. Palgrave Macmillan, second edition.

[14] Stroud, A., and D. Secrest (1966), *Gaussian Quadrature Formulas*. Englewood Cliffs, NJ: Prentice Hall.