Trinomial-tree based parallel option price valuations

Alexandros V. Gerbessiotis

Department of Computer Science
New Jersey Institute of Technology
Newark, NJ 07102

Research Report: CS-02-03

Trinomial-tree based parallel option price valuations

Alexandros V. Gerbessiotis

CS Department

New Jersey Institute of Technology

Newark, NJ 07102

USA.

June 25, 2002

Abstract

We examine how trinomial-tree based computations such as those involved in American or European-style option price valuations can be performed in parallel. Towards this we introduce a parallel algorithm for performing such computations on trinomial trees. The algorithm is described and analyzed in an architecture independent setting and achieves optimal theoretical speedup $O(p)$ and is thus within a $1+o(1)$ multiplicative factor of the corresponding sequential method. We verify the practicality and plausibility of the designed algorithm by carrying out an experimental study of an implementation of the algorithm on a high-latency parallel system, a cluster of PC workstations. The algorithmic and programming methodology used to design and analyze the algorithm allows its implementation to work with only recompilation of the source code under two parallel programming libraries: MPI (LAM-MPI) and BSPlib thus making the implementation not only architecture but also communication-library independent.

Keywords: BSP model, trinomial trees, option valuations

1 Introduction

We investigate in this work how trinomial-tree based option price valuations of assets such as American-style stock options can be performed in parallel. We introduce a method in performing these valuations in parallel in optimal space and in parallel time that is also optimal, thus achieving optimal parallel speedup. We design and analyze the proposed algorithm using the parallel programming methodology offered by the Bulk-Synchronous Parallel (BSP) model of computation, which deals explicitly with the notion of communication and synchronization among parallel computational threads. The BSP model thus allows the design, analysis and programming of parallel algorithms and programs in an architecture independent setting. In order to validate the theoretical conclusions drawn from the analysis of the proposed algorithm we also perform an experimental study of an implementation of the proposed algorithm on a high-latency parallel platform consisting of a cluster of dual processor PC workstations connected by a high performance 100Mbit ethernet switch. The experimental results obtained seem to validate the usefulness and usability of the BSP model in measuring parallel computation and communication performance.

The *Bulk-Synchronous Parallel* (BSP) model of computation has been proposed by L.G. Valiant [19, 20] as a unified framework for the design, analysis and programming of general purpose parallel computing systems. It offers the prospect of achieving both scalable parallel performance and architecture independent parallel software, and provides a framework which permits the performance of parallel and distributed systems to be analyzed and predicted in a precise way. The term *bulk-synchronous* reflects the underlying position of the model between the two extremes, (i) entirely synchronous systems (a global clock is utilized to achieve synchronization on every instruction), and (ii) fully asynchronous systems (synchronization is realized pairwise whenever communication is required). Processor components in the BSP model advance jointly through the program, with the required remote communication occurring between *supersteps*. The model consists of the following three components. (i) A *collection of p processor/memory components* numbered $0, \ldots, p-1$. It is assumed that each processing unit consists of a sequential processor with a block of local memory. (ii) A *communication network* that can deliver messages point to point among the components. (iii) A *mechanism* for efficient synchronization, in *barrier style*, of all or a subset of the components.

The performance of any BSP computer can be characterized by the following three parameters: (i) p - the number of processor components, (ii) L - the minimum elapsed time, measured in basic computational operations, between successive synchronization operations, and (iii) g - the cost of communication per word (inverse of router throughput). More formally, g is defined so that the time required to realize an h-relation in continuous message usage (i.e. for large enough h) is gh times steps; for small h the cost of communication is L (latency bound). Computation on the BSP model proceeds in a succession of *supersteps*. During a superstep each processing unit is given a task to perform using data already available locally before the start of the superstep. The task may include (i) computation steps on values held locally at the start of the superstep, (ii) message transmissions, and (iii) message receipts. A superstep can complete at any time after the first L time units. It is noted that, although BSP stresses global barrier style synchronization, pairs of processor components may synchronize pairwise by sending messages to and from an agreed memory location. The time complexity of a superstep S in a BSP algorithm is determined as follows. Each superstep is charged $\max\{L, x + gh\}$ basic time steps, where x is the maximum number of basic computational operations executed by any processor during S, and h is the maximum number of messages transmitted or received by any processor. Note that, if h_t is the maximum number of messages transmitted by any processor during S, and h_r is the maximum number of messages received by any processor during S, then $h = \max\{h_t, h_r\}$.

We specify the performance of a BSP algorithm $\mathcal{A}$ in three parts. First, we specify a sequential algorithm $\mathcal{A}^*$ with which we are comparing it and make explicit how we charge for the basic operations of both $\mathcal{A}$ and $\mathcal{A}^*$. Second, we specify two ratios π and μ. The former, π, is the ratio between the computation time $C_{\mathcal{A}}$, of the BSP algorithm, over the time $C_{\mathcal{A}^*}$ of the corresponding sequential algorithm divided by p, i.e., $\pi = pC_{\mathcal{A}}/C_{\mathcal{A}^*}$. The latter, μ, is the ratio between the parallel time $M_{\mathcal{A}}$ required by the communication supersteps of the BSP algorithm and the computation time of $\mathcal{A}^*$ divided by p, i.e., $\mu = pM_{\mathcal{A}}/C_{\mathcal{A}^*}$. When we specify μ it is necessary to be explicit about the amount of information that can be conveyed in a single message. Finally, we specify conditions on n, p, L and g that are sufficient for the algorithm to make sense and the claimed bounds on π and μ to hold. We also claim as corollaries some sufficient conditions for the most interesting optimality

criteria, such as c-optimality, i.e., $\pi = c + o(1)$ and $\mu = o(1)$. All asymptotic bounds refer to the problem size as $n \to \infty$.

2 Trinomial-tree based option price valuations

The trinomial-tree model was introduced in [3] for pricing primarily American-style and European-style options [14, 4] on a single underlying asset although they can also be used to price some derivatives on two correlated assets [14]. Pricing options under the Black-Scholes model [2] requires the solution of the partial differential equation satisfied by the option price. Option prices could be obtained by building a discrete time and state binomial model of the asset price and then applying discounted expectations [4]. A generalization of such a binomial valuation model [18] to a trinomial model and option valuations on the trinomial model are useful since solving the partial differential equation of the option price by the explicit finite difference method [17] is equivalent to performing discounted expectations in a trinomial-tree [4]. Because the asset price in a trinomial-tree moves in three directions compared with only two for a binomial tree the time horizon (number of steps) can be reduced in a trinomial-tree to attain the same accuracy obtained by a binomial-tree. There are various ways one can choose the jump size and move probabilities of a trinomial-tree.

For an asset whose current price is S, and r is the riskless and continuously compounded interest rate, the stochastic differential equation for the risk-neutral geometric Brownian motion (GBM) model of an asset price paying a continuous dividend yield of δ per annum [4] is given by the following formula

$$dS = (r - \delta)Sdt + \sigma S dz. \tag{1}$$

Since it is more convenient to work in terms of $x = \ln S$, we use

$$dx = vdt + \sigma dz, \quad \text{where} \quad v = r - \delta - \sigma^2/2. \tag{2}$$

In a trinomial model of the asset price, over a small time interval Δt the asset price can go up by Δx, stay the same, or go down by Δx, with probabilities p_u, p_m and p_d respectively. Δx is called

the space step of the process. This movement of the asset price is depicted in two different ways in Figure 1.

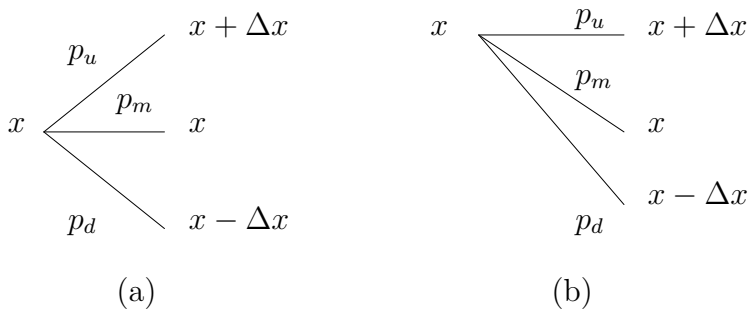

Figure 1: Trinomial tree model: two views.

View (a) in Figure 1 is the traditional way to view the trinomial process. View (b), however, is the one that we will adapt in this work as it will facilitate the exposition of the computations performed in the introduced algorithm. The drift and volatility parameters of the continuous time process are captured by Δx, p_u, p_m, p_d and Δx is chosen so that $\Delta x = \sigma\sqrt{3\Delta t}$ (see [4]). The discretization of the GBM requires that the drift and volatility parameters match the first two moments (mean and variance) of the distribution and the probabilities add up to one. Therefore,

$$E[\Delta x] = p_u \cdot \Delta x + p_m \cdot 0 + p_d \cdot (-\Delta x) = v\Delta t, \tag{3}$$

$$E[\Delta x^2] = p_u \cdot \Delta x^2 + p_m \cdot 0 + p_d \cdot (\Delta x^2) = \sigma^2\Delta t + v^2\Delta t^2, \tag{4}$$

$$p_u + p_m + p_d = 1. \tag{5}$$

Solving equations (3), (4), and (5) yields the following solutions for p_u, p_m and p_d.

$$p_u = \frac{1}{2}\left(\frac{\sigma^2\Delta t + v^2\Delta t^2}{\Delta x^2} + \frac{v\Delta t}{\Delta x}\right) \tag{6}$$

$$p_m = 1 - \frac{\sigma^2\Delta t + v^2\Delta t^2}{\Delta x^2}. \tag{7}$$

$$p_d = \frac{1}{2}\left(\frac{\sigma^2\Delta t + v^2\Delta t^2}{\Delta x^2} - \frac{v\Delta t}{\Delta x}\right) \tag{8}$$

The trinomial process of Figure 1 can then be repeated a number of times to form a trinomial-tree such as the one depicted in Figure 2 in two different ways. Figure 2 depicts a trinomial-tree for $n = 4$ time steps, i.e. of horizon $n = 4$. The number of levels in such a tree is $n + 1 = 5$ and the number of nodes at any level i is equal to $2i + 1$. A node in the tree is indexed by a pair (i, j), where i indicates the level (column index) and j the distance from the top (row index). Time t can then be inferred from the level index i: $t = i\Delta t$. The standard view of a trinomial-tree is Figure 2 (a). In the standard view upward and downward moves point upwards and downwards to the right respectively, and steady asset moves point horizontally to the right. In this work we use the alternative view depicted in Figure 2(b). In the alternative view upward moves point to the right horizontally and downward moves point steeply downards to the right, whereas steady moves point downwards to the right but at a less steeper angle. In such a view, node (i, j) is thus connected to node $(i + 1, j)$ (upward move), to node $(i + 1, j + 1)$ (steady move), and to node $(i + 1, j + 2)$ (downward move). Row j also contains $n + 1 - j$ nodes in the alternate view.

The option price and the asset price at node (i, j) are given by $C[i, j] = C_{i,j}$ and $S[i, j] = S_{i,j}$ respectively. The latter is straightforward to compute from the number of up and down moves required to reach (i, j) from $(0, 0)$ and is given by the following expression for the alternative view of Figure 2(b).

$$S[i, j] = S[0, 0] \cdot exp((i - j) \cdot \Delta x). \tag{9}$$

Time step n corresponds to maturity, i.e. $T = n\Delta t$. The option prices at maturity (i.e. time step n) are determined by the pay off and thus for a call option (intent to buy the asset at a predetermined strike price) this is $C_{n,j} = \max(0, S_{n,j} - K)$ and for a put option (intent to sell) this is $C_{n,j} = \max(0, K - S_{n,j})$. K is the strike price at maturity $T = n\Delta t$ for a European-style option, and the strike price at any time before or on maturity for an American-style option.

Then one can compute option prices as discounted expectations under a risk-neutral assumption, and thus option prices at earlier nodes can be computed as discounted expectations. For an

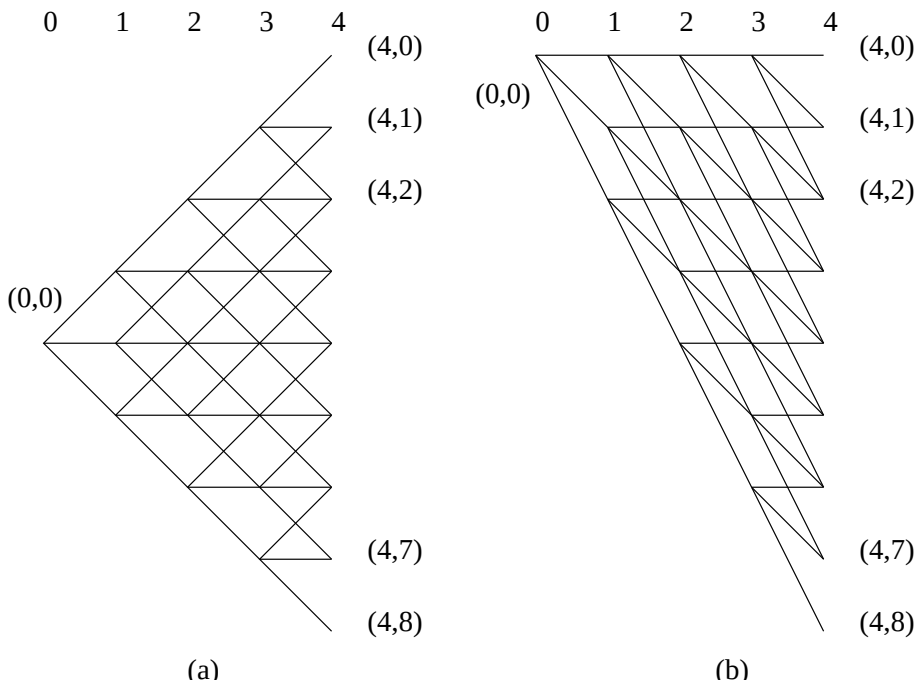

Figure 2: A Trinomial Tree: two views for $n = 4$

American put option (that can be exercised at any time before or on maturity), for $i < n$,

$$C_{i,j} = \max\left(e^{-r\Delta t}(p_u C_{i+1,j} + p_m C_{i+1,j+1} + p_d C_{i+1,j+2}), K - S_{i,j}\right) \tag{10}$$

(for an American call option use $S_{i,j} - K$ instead of $K - S_{i,j}$), whereas for a European call option (that can be exercised on maturity only), for $i < n$,

$$C_{i,j} = e^{-r\Delta t}(p_u C_{i+1,j} + p_m C_{i+1,j+1} + p_d C_{i+1,j+2}). \tag{11}$$

The expression for $C_{n,j}$ along with equation (9), and equation (10) or (11) can be used to compute the option price at every time step and node of the trinomial-tree. The option price at the start is thus $C_{0,0}$.

Figure 3 provides pseudocode for valuating the nodes of a trinomial-tree for American-style put options.

begin AMERICAN_PUT (K, S, n, T, σ,r,δ)

1. $\Delta t = T/N$;$v = r - \delta - \sigma^2/2$;
2. $\Delta x = \sigma\sqrt{3dt}$;
3. $p_u = \frac{1}{2}\left(\frac{\sigma^2\Delta t + v^2\Delta t^2}{\Delta x^2} + \frac{v\Delta t}{\Delta x}\right)$;
4. $p_m = 1 - \frac{\sigma^2\Delta t + v^2\Delta t^2}{\Delta x^2}$;
5. $p_d = \frac{1}{2}\left(\frac{\sigma^2\Delta t + v^2\Delta t^2}{\Delta x^2} - \frac{v\Delta t}{\Delta x}\right)$;
6. $discount = exp(-r\Delta t)$;
7. $S[0] = S * exp(n * \Delta x)$; //$S[0..2n]$: asset prices
8. **for** $j = 1$ **to** $2 * n$ **do**
9. $S[j] = S[j-1] * exp(-\Delta x))$; // asset price at maturity
10. **for** $j = 0$ **to** $2 * n$ **do** //$C[0..2n]$: option prices
11. $C[j] = \max(0, K - S[j])$; // option price at maturity
12. **for** $i = n - 1$ **downto** 0 **do**
13. **for** $j = 0$ **to** $2 * i$ **do**
14. $C[j] = discount * (p_u * C[j] + p_m * C[j+1] + p_d * C[j+2])$;
15. $S[j] = S[j] * exp(-\Delta x)$;
16. $C[j] = \max(C[j], K - S[j])$;
17. **return** ($C[0]$);

end AMERICAN_PUT

Figure 3: Procedure AMERICAN_PUT.

3 Trinomial-tree based valuations in parallel

A trinomial-tree based valuation with horizon n has a total of $(n+1)^2$ nodes, as level (time-step i) consists of $2i+1$ nodes. Depending on the application, the option price at node $(0,0)$ may only be required; in some other cases option prices on all nodes need to become available at the conclusion of the valuation. In the former case the maintenance in main memory of the information stored in all the nodes of the tree may be redundant and only linear in n memory space needs to be allocated. The pseudocode in Figure (3) reflects this assumption. Otherwise the one dimensional arrays $S[j]$ and $C[j]$ can be replaced by two dimensional arrays $S[i,j]$ and $C[i,j]$. Lines 9 and 11 then involve

$S[n, j]$ and $C[n, j]$ calculations and lines 14-17 involve $C[i, j]$, $S[i, j]$ calculations. If all intermediate results need to be maintained, the total space requirements become quadratic in n. We deal with the former case in this work. The running time claims, however, apply unchanged to either case. Information related to option price is maintained in array C and asset price in array S by allocating space proportional to $2n + 1$, where n is the time horizon.

In a parallel computer, we distribute both arrays evenly among the p processors in the following way. We introduce parameter b that will be called the block factor of the memory representation of C (or S). We store C (or S) in a blocked wrapped-mapping distribution such that the first b elements of C are stored in processor 0, the next b available elements to processor 1, and so on, the next b elements to p rocessor $p - 1$, and wrapping around the processors, the next b elements to processor 0, and so on. The last processor in this assignment may thus be assigned fewer than b elements.

Under this scheme, element i of the array is assigned to processor $\lfloor i/b \rfloor \% p$, where p is the number of processors available. Within that processor, element i is the $(\lfloor i/(pb) \rfloor * b + (i\%b))$-th element of the local copy of the array. Given this partitioning scheme the maximum imbalance between any two processors is b. In the introduced algorithm, parameter b will be kept small i.e. $b = o(n/p)$, although b will also be chosen so that $b = \omega(1)$.

The parallel algorithm for trinomial-tree based option price valuations distinguishes two cases depending on whether b is even or odd. For the sake of an example, Figures 4 and 5 depict the order the valuations are performed on a $b \times b$ block defined by the black-filled nodes of a trinomial-tree, where $b = 5$ and $b = 6$ respectively. For the black-filled nodes only are node dependencies shown in detail and in full in Figures 4 and 5. Let some processor, call it without loss of generality zero, store the $b \times b$ block of black-filled nodes. Let processor one, the next processor in the linear processor ordering, store the empty-filled nodes of the following b rows of the trinomial tree partially depicted in Figures 4 and 5. Nodes of levels before and after the depicted levels and on rows before or after the depicted rows are not shown, nor are the full dependencies of the empty-filled nodes shown.

In a parallel valuation of a trinomial-tree the parallel algorithm walks through the levels backwards from n down to 0, by processing b levels of the tree simultaneously in a single update. This

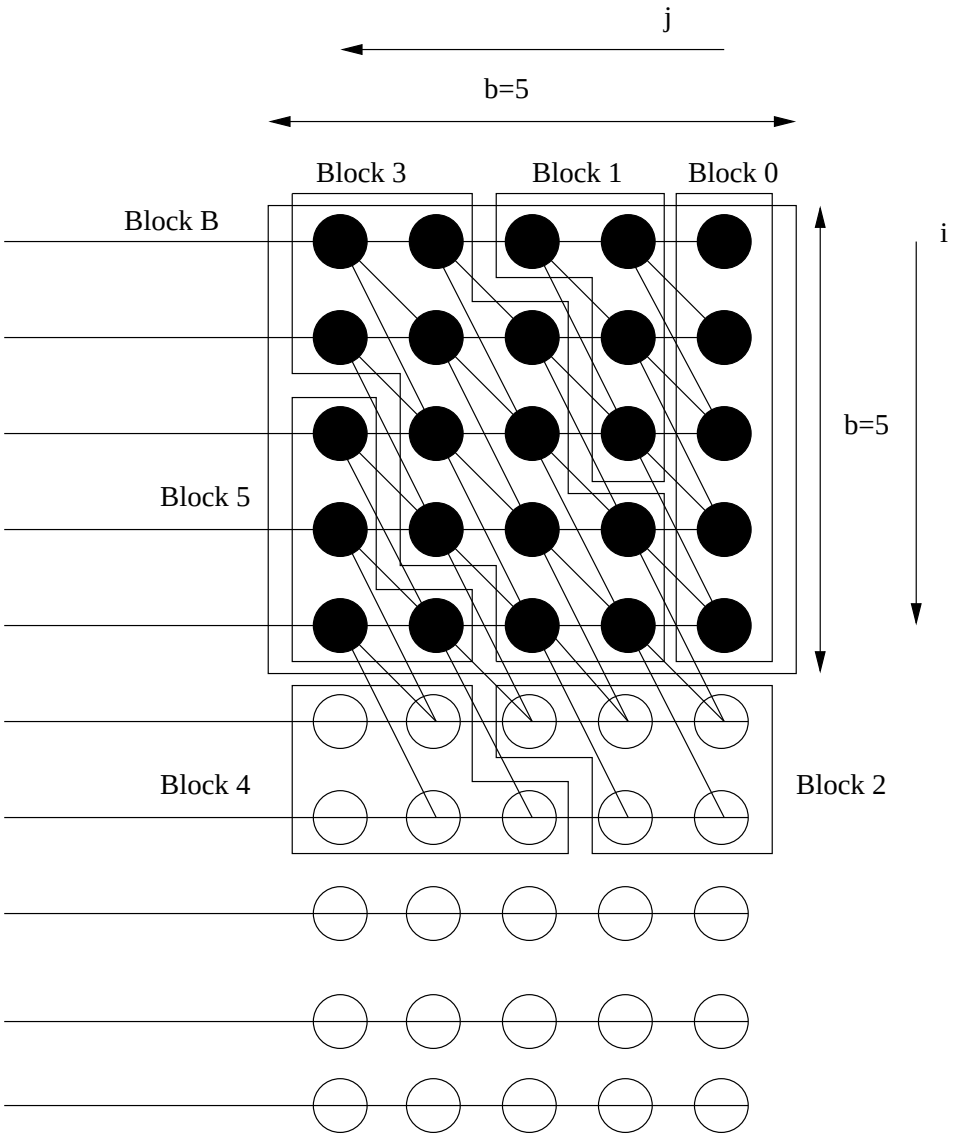

Figure 4: Trinomial tree parallel valuation scheduling; b is odd ($b = 5$)

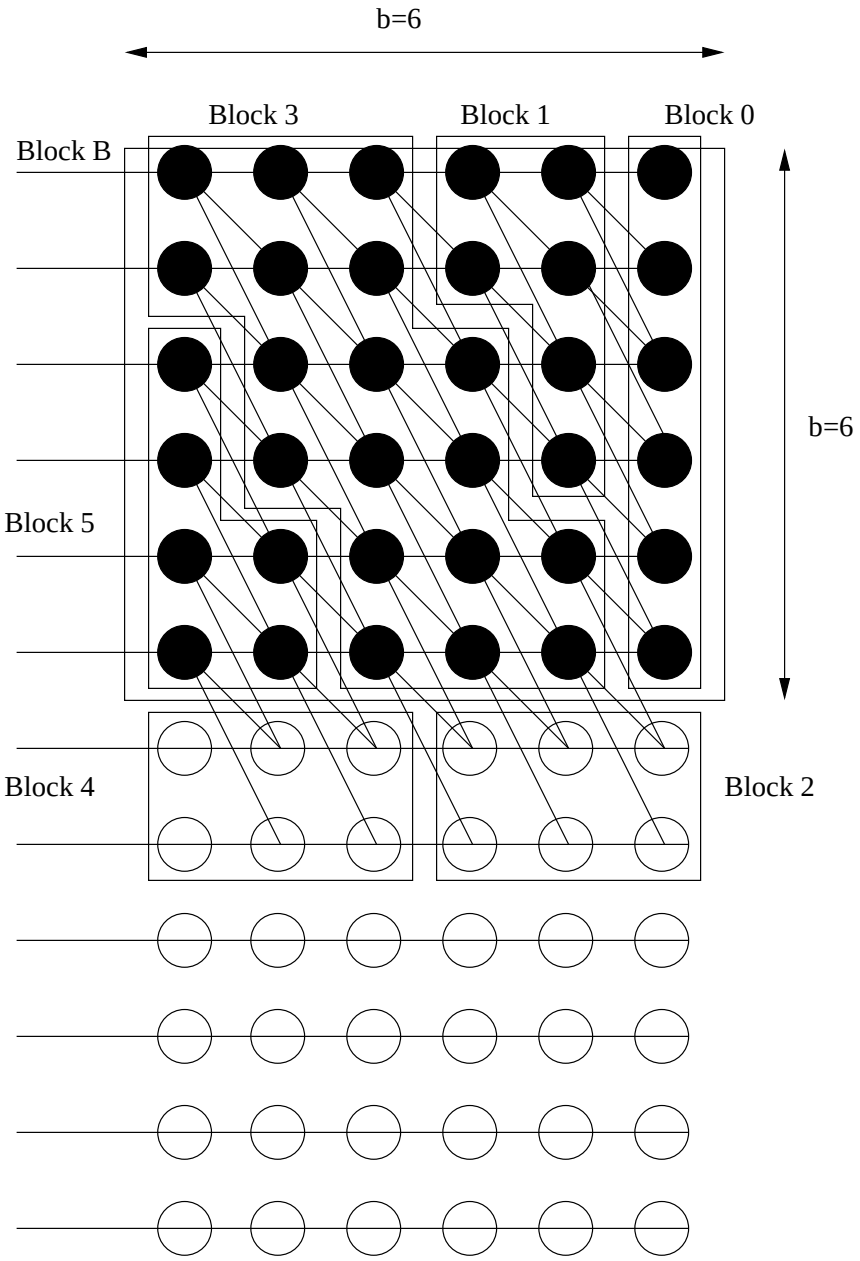

Figure 5: Trinomial tree parallel valuation scheduling; b is even ($b = 6$)

is done to avoid frequent communication and synchronization operations. In fact the algorithm performs three supersteps two of which include communication for every b levels of the tree. The memory allocation scheme that subdivides the tree into square blocks of $b \times b$ nodes allows each processor to perform various updates blockwise; each processor maintains in its memory multiple such blocks. In the last few levels (close to the level 0 origin) a processor may hold a single block, and every iteration one processor also holds an incomplete (upper diagonal block) involving the last rows of the currently examined b levels.

We call the black-filled nodes of a $b \times b$ node block whose nodes will be valuated Block B. This Block B of nodes is subdivided into four blocks of nodes: Block 0, 1, 3, and 5; Blocks 1 and 5 are symmetric. An invariant that will be maintained in the parallel algorithm is that Block 0 nodes have already been valuated. This is so because in the boundary (initial) case, Block 0 nodes consist of nodes of level n and these nodes are evaluated separately in lines 7-11 in Figure 3 before the main loop of lines 12-13 that valuates the remaining nodes; this preprocessing step is retained in the parallel algorithm.

Among the remaining blocks of Block B, Block 1 nodes can be valuated using information locally and already available in Block B as these nodes need only get the option prices of the Block 0 nodes. Block 3 and 5 nodes stored in processor zero, however require non-local information that is available to the next processor, processor one according to our example of Figures 4 and 5.

The valuation of Block 3 nodes in processor zero is dependent on the option prices of the Block 2 nodes, and the valuation of Block 5 nodes in processor zero is dependent on the option prices of the Block 4 nodes. Block 2 nodes and Block 4 nodes are white-filled and thus belong by assumption to processor one. Therefore Block 2 and 4 option prices need to be communicated from processor one to processor zero as soon as they become available (i.e. as soon as the nodes are valuated). Block 2 white-filled nodes are themselves Block 1 and Block 0 to processor one (not zero), whereas Block 4 white-filled nodes are Block 3 nodes to processor one. Therefore option prices for Block 2 nodes can be communicated as soon as Block 1 and Block 0 nodes have been valuated in parallel in all participating processors; Block 4 node option prices can be communicated as soon as Block 3 nodes have been valuated. Block 2 and Block 4 have the same number of nodes; their shapes may

differ depending on whether b is odd or even.

Therefore for the Block B node valuations the work schedule performed by an arbitrary processor $(m-1+p)\%p$ is listed below. Given that Block 0 nodes have already been valuated the algorithm works on b levels but valuates nodes of $b-1$ levels at a time.

i. Block 0 nodes in processor $m-1+p\%p$ have already been valuated.

ii. Block 1 nodes in processor $m-1+p\%p$ and in parallel with other processors are valuated using the option prices of Block 0 nodes.

iii. Block 2 nodes are Block 0 and Block 1 nodes in processor m, and have thus been valuated in (i) and (ii). Option prices of Block 2 nodes are communicated to processor $m-1+p\%p$ from processor m.

iv. Block 3 nodes in processor $m-1+p\%p$ and in parallel with other processors are then valuated.

v. Block 4 nodes are Block 3 in processor m, and have thus been valuated in (iv). Option prices of Block 4 nodes are communicated to processor $m-1+p\%p$ from processor m.

vi. Block 5 nodes in processor $m-1+p\%p$ and in parallel with other processors are then valuated.

Steps (ii)-(vi) are repeated until the nodes of all levels of the tree are valuated. It is clear that steps (ii) and (iii) form the first superstep, (iv) and (v) the second superstep, and (vi) the third superstep. The first and second supersteps involve both computation (option price valuations) and communication, whereas the third superstep is pure computation. The following pseudocode for BSP_Trinomial describes the major steps of a trinomial-tree based option price parallel valuation. All references to lines of code are to the code of American_Put depicted in Figure 3.

In order the collectively describe the various groups of nodes, let us index the nodes of Block B using indices $0 \leq i < b$, $0 \leq j < b$, where j indexes the columns (levels of the tree) right to left and i indexes the rows (rows of the tree) of Block B top to bottom as shown in Figure 4. Block 1 nodes are nodes (i,j) such that $1 \leq j \leq \lfloor (b+1)/2 \rfloor - 1$, and $0 \leq i < b-2j$. Block 2 nodes consist of nodes (i,j) such that $0 \leq j < \lfloor b/2 \rfloor$ and $0 \leq i \leq 1$; in addition, if b is odd Block 2 also includes

begin BSP_TRINOMIAL ()

// Note: We use a two dimensional array $C_{i,j}$ to refer to the

// option price of the node in row j of time step i

1. Initialize level n; //i.e. assign $C_{n,j}$ prices to Block 0 nodes
2. **for (** $k = n; k > 0; k = k - b + 1$)

//First Superstep (Steps 3 and 4)

3. Perform the computation of Lines 12-16 for Block 1 nodes;
4. Communicate $C_{i,j}$ values for Block 2 nodes from every processor m to processor $m - 1 + p\%p$;

//Second Superstep (Steps 5 and 6)

5. Perform the computation of Lines 12-16 for Block 3 nodes using the information received in Step 4;
6. Communicate $C_{i,j}$ values for Block 4 nodes from every processor m to processor $m - 1 + p\%p$;

//Third Superstep (Step 7)

7. Perform the computation of Lines 12-16 for Block 5 nodes using the information received in Steps 4 and 6;

//Option prices for levels $k - b + 1, \ldots, k$ have been valuated.

end BSP_TRINOMIAL

Figure 6: Procedure BSP_TRINOMIAL.

node $(0, b/2)$. Block 3 nodes can be described as the union of two groups: nodes (i, j) such that $1 \leq j \leq \lfloor (b+1)/2 \rfloor - 1$ and $b - 2j \leq i < b$ and nodes (i, j) such that $\lfloor (b+1)/2 \rfloor \leq j < b$ and $0 \leq i < b - 2(j - \lfloor (b+1)/2 \rfloor) - odd(b)$, where odd$(b)$ is 1 if b is odd and 0 otherwise. Block 4 nodes consist of nodes (i, j) such that $\lfloor (b+1)/2 \rfloor \leq j < b$ and $0 \leq j \leq 1$; in addition, if b is odd Block 2 also includes node $(1, b/2)$. Finally Block 5 consists of nodes (i, j) such that $\lfloor (b+1)/2 \rfloor \leq j < b$ and $b - 2(j - \lfloor (b+1)/2 \rfloor) - odd(b) \leq i < b$.

3.1 Time analysis of algorithm BSP_Trinomial

With reference to Figure 6 for algorithm BSP_Trinomial, we describe the space and time requirements of the parallelization of the valuation method described in Figure 3. Note that C becomes a two-dimensional array in Figure 6 to facilitate exposition only.

Step 1. The $2n+1$ nodes of the trinomial-tree at time n are split into $\lceil(2n+1)/b\rceil$ groups of size b, each processor getting at most $\lceil\lceil(2n+1)/b\rceil/p\rceil$ groups. Each group consists of at most b nodes. The initialization step of Step 1 of Figure 6 thus requires time proportional to $\lceil\lceil(2n+1)/b\rceil/p\rceil b \leq ((2n+1)/(pb)+1)b = (2n+1)/p + b$.

Loop of Steps 2-8. The loop of Step 2 starts at $k = n$, and every time it is executed, all the nodes of the tree of levels $k-b+1 \ldots k-1$ are valuated. We call the valuation of all the nodes of the trinomial-tree in a single iteration of Step 2, an update. For any value of k, the i-th update, where $i = \lfloor(n-k+(b-1))/(b-1)\rfloor$, thus involves levels $k-b+1 \ldots k-1$. Eventually k reaches zero or a negative value and the loop terminates. The total number of updates performed until k reaches 0 (or the number of times the loop of step 2 is executed) is $\lceil n/(b-1)\rceil$. In calculations later in this section, for simplicity and without loss of generality, we will assume that n is a multiple of $b-1$.

Steps 3, 5, and 7. Each processor is assigned a number of square $b \times b$ blocks of the trinomial-tree, except for the last block of the last processor that is an upper diagonal array/block. In the update that includes the k-th time step (i.e. in the i-th update) since the number of nodes of the trinomial-tree at step k is $2k+1$ we obtain that each one of the p processors will valuate at most $\lceil\lceil(2n-2(i-1)(b-1)+1)/b\rceil/p\rceil$ blocks of dimension $b \times b$. On each such block $b(b-1)$ node valuations are performed on the nodes of Blocks 1, 3, and 5 thus excluding the b nodes of Block 0. The total work required for the i-th update is thus equal to $\lceil\lceil(2n-2(i-1)(b-1)+1)/b\rceil/p\rceil b(b-1)$ per processor. In the remainder we are going to make the assumption that n is a multiple of $b-1$, an assumption that does not affect the asymptotics of the calculations below. Steps 3 and 4, 5 and 6, and 7 form three individual supersteps. The parallel computation time involved in steps 3, 5 and 7 is estimated immediately below. After that the parallel communication involved in step 4 and 6 is computed; the synchronization periodicity contributions of the first two supersteps are

also included there. The synchronization periodicity contribution of the third superstep is finally included. Let $N = \lceil \frac{n}{(b-1)} \rceil$. The parallel computation time of steps 3, 5, and 7 is thus given by the following expression.

$$\begin{aligned}
\sum_{i=1}^{N} \left\lceil \frac{\left\lceil \frac{2n+1-2(i-1)(b-1)}{b} \right\rceil}{p} \right\rceil b(b-1) &= \sum_{i=1}^{N} \left(\frac{2n+1-2(i-1)(b-1)}{bp} + 1 \right) b(b-1) \\
&= \sum_{i=1}^{N} \left(\frac{(2n-1)(b-1)}{p} - \frac{2i(b-1)^2}{p} + b(b-1) + \frac{2b(b-1)}{p} \right) \\
&= \frac{(2n-1)(b-1)}{p} N + b(b-1)N + \frac{2b(b-1)}{p} N - \\
&\quad \frac{2(b-1)^2}{2p} N(N+1) \\
&= \frac{(2n-1)n}{p} + nb + \frac{2bn}{p} - \frac{n^2}{p} - \frac{(b-1)n}{p} \\
&= \frac{n^2}{p} + nb + \frac{nb}{p}. \qquad (12)
\end{aligned}$$

Steps 4 and 6. The communication performed in step 4 and 6 requires the communication of b option prices per $b \times b$ block as the size of Block 2 is equal to b and equal to the size of Block 4 (note: out of the b nodes of Block 4 only $b-2$ are really needed). Therefore the communication and synchronization time for steps 4 and 6 and the first two supersteps is bounded above by the following expression.

$$\begin{aligned}
\sum_{i=1}^{N} 2 \max \left\{ L, \left\lceil \frac{\left\lceil \frac{2n+1-2(i-1)(b-1)}{b} \right\rceil}{p} \right\rceil bg \right\} &= \sum_{i=1}^{N} 2 \max \left\{ L, gb \cdot \left(\frac{2n-1-2i(b-1)+2b}{bp} + 1 \right) \right\} \\
&= 2LN + \\
&\quad \sum_{i=1}^{N} 2g \cdot \left(\frac{(2n-1)}{p} - \frac{2i(b-1)}{p} + \frac{2b}{p} + b \right) \\
&= 2LN + 2g \frac{(2n-1)}{p} N + 2g \frac{2b}{p} N + 2gbN + \\
&\quad -2g \frac{(b-1)}{p} N(N+1) \\
&= \frac{2Ln}{(b-1)} + 2g \frac{n^2}{p(b-1)} + 2gn \frac{b}{b-1} + 2g \frac{nb}{p(b-1)}. \qquad (13)
\end{aligned}$$

The total parallel computation time results by combining the time for step 1 with that for steps 3, 5, and 7.

$$C(n,p) = \frac{n^2}{p} + nb + \frac{nb}{p} + \frac{2n+1}{p} + b = \frac{(n+1)^2}{p} + \frac{n}{2p} + (n+1)b. \quad (14)$$

The synchronization and communication time is that reported for steps 4 and 6 plus a term LN reflecting the contributions of the third superstep.

$$M(n,p,L,g) = \frac{3Ln}{(b-1)} + 2g\frac{n^2}{p(b-1)} + 2gn\frac{b}{b-1} + 2g\frac{nb}{p(b-1)}. \quad (15)$$

We note that the sequential computation time is given by $T(n) = \sum_{i=0}^{n}(2i+1) = (n+1)^2$. The following Theorem can then be derived.

Theorem 1 *For any integer n, $p \leq n$, $b \geq 1$ such that $b-1$ divides n, algorithm* BSP_Trinomial *for option-price valuations on a trinomial-tree requires parallel computation time equal to*

$$C(n,p) = \frac{(n+1)^2}{p} + \frac{n}{2p} + (n+1)b,$$

and parallel communication (and synchronization) time

$$M(n,p,L,g) = 3L\frac{n}{(b-1)} + 2g\frac{n^2}{p(b-1)} + 2gn\frac{b}{b-1} + 2g\frac{nb}{p(b-1)}.$$

One-optimality for BSP_Trinomial can be achieved for a wide range of the BSP parameters L and g. The following corollary describes such a case.

Corollary 1 *For any integer n, $p \leq n$, $b \geq 2$ such that $b-1$ divides n, algorithm* BSP_Trinomial *for $b = o(n/p)$, $L = o(nb/p)$, and $g = o(b)$ achieves $\pi = 1 + o(1)$ and $\mu = o(1)$.*

Proof. For $Ln/(b-1) = o(n^2/p)$ synchronization periodicity costs are absorbed in the low order terms of the running time. Therefore for $L = o(nb/p)$, such costs become insignificant. In computation time, the overhead of the blocked allocation of nodes to processors is reflected in the contributing term $(n+1)b$. This term becomes insignificant for $(n+1)b = o(n^2/p)$, i.e. for $b = o(n/p)$. The contributions of communication time become negligible as long as $g = o(b)$. By

dividing either $C(n,p)$ or $M(n,p,L,g)$ by sequential running time $T(n)$ we deduce that for the given conditions of L, b and g we have that $\pi = 1 + O(1/n) + O(pb/n)$ and $\mu = o(1)$. The corollary then follows. ■

4 Performance Evaluation

As the BSP model is not just an abstract architectural model, it may also be employed as a programming paradigm. This way, a BSP program can be viewed as a succession of supersteps, with the required non-local communication occurring at the end of each superstep and can also be realized as a library of functions with architecture independent semantics for process creation, remote data access and bulk synchronization. The Oxford BSP Toolset, *BSPlib*, implements such a paradigm [15] and provides library support for BSP programming. The library functions of the Oxford BSP Toolset are callable from standard imperative languages such as C and Fortran. The effort required to learn the basics of *BSPlib* is minimal as it consists of about 10-15 functions. BSP-model programming support can also be made available by other non BSP-specific libraries such as MPI (Message Passing Interface) that may also support the simple communication and synchronization primitives required for programming under the BSP model. Lam-MPI is such an example. We have thus made our BSP code that implements BSP_Trinomial run transparently not only on a BSP-specific library (i.e. *BSPlib*) but also on MPI, a non BSP-specific library, through the LAM-MPI [16] implementation of MPI. We have written our code in an architecture and library independent way using standard ANSI C and few generic communication primitives/functions related to processor creation, destruction, identification, and direct memory interprocessor communication. These generic communication primitives are then translated at compile-time into specific function calls of one or the other of the two libraries. For example, if *BSPlib* is chosen, direct memory communication is performed through the bsp_put, bsp_get and similar calls of *BSPlib*. If LAM-MPI is chosen instead, direct memory communication is performed through the corresponding operations MPI_Put, MPI_Get that are part of the MPI-2 extensions which are supported by LAM-MPI. No change in the source code is required to make the program run under one or the other library. Therefore we can claim that

our code is not only *architecture independent* but also *parallel programming/communication library independent*.

The implementation language is ANSI C and the code was tested for scalability and portability on a cluster of 16 dual-processor PC workstations (PentiumII 350Mhz, Redhat Linux 7.1-running, 128MB RAM equipped) with communication performed through 100Mbit 3Com-905B Ethernet cards and the workstations connected by a 24-port CISCO Catalyst 2924M-XL-EN switch. The default GNU Project's gcc compiler is used through either the *BSPlib* or the LAM-MPI front-end, and the source-code is compiled with the `-O3` compiler option set. Timing is obtained through the use of the real-time (wall-clock time) clock functions available in both libraries (`bsp_time` of *BSPlib* [13] and `MPI_Wtime` of MPI/LAM-MPI). All results and variables used for communication and computation involve ANSI C `double` 64-bit data types. The code developed for this experimental study is publically available at the author's Web-page [7].

The timing results obtained are discussed later in this section. The tested platform is either the 16-node PC cluster utilizing only one CPU per node, or an 8-node (or the full 16-node cluster) subcluster utilizing both CPUs. By setting up the cluster this way we also examined the effects of symmetric multiprocessing. The reported speedup results reflect valuations of American-style put options; a report on the performance and relative merits of the implementation of BSP_Trinomial is also included. Timing and/or speedup figures are in general truncated to two or fewer decimal digits. We performed experiments for problem sizes $n = 32768, 65536$. We also varied block size b; in the experiments we set $b = 512, 640, 786, 1024$; note that since $b - 1$ does not divide n, one extra update is required. For $p = 1$ timing results are reported, which are independent of b. We used a sequential implementation to derive these timing figures; we did not run the parallel implementation run on a single processor. For processor sizes $p = 4, 8, 16, 32$, relative speedup figures are only reported rather than actual timing results; it is straightforward to obtain time from these speedup figures and the $p = 1$ timing results, if necessary.

Table 1 shows timing/speedup results for problem sizes $n = 32768$ and $n = 65536$ on the 16-node PC cluster. Table 2 shows timing/speedup results for problem sizes $n = 32768$ and $n = 65536$ on using both processors of every SMP node.

	$n = 32768$				$n = 65536$			
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
512	148.96	2.44	4.62	7.65	663.98	2.68	5.17	9.31
640	148.96	2.48	4.77	7.83	663.98	2.80	5.32	9.59
786	148.96	2.48	4.77	7.75	663.98	2.80	5.33	9.58
1024	148.96	2.40	4.62	6.91	663.98	2.72	5.33	8.69

Table 1: Execution Time and Speedup using BSPlib on a 16-node PC cluster.

	$n = 32768$					$n = 65536$				
	Time	Speedup				Time	Speedup			
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
512	148.96	2.40	4.75	7.94	11.03	663.98	2.62	5.23	9.70	13.40
640	148.96	2.46	4.76	8.01	11.63	663.98	2.74	5.40	9.83	14.56
786	148.96	2.49	4.76	8.02	11.39	663.98	2.75	5.42	9.79	14.76
1024	148.96	2.40	4.62	7.88	11.45	663.98	2.74	5.34	9.78	15.52

Table 2: Execution Time and Speedup using BSPlib on an SMP cluster.

Table 1 and Table 2 indicate the increasing performance of BSP_Trinomial for increasing problem size. The best performance of the algorithm is obtained when block factor b is between $b = 640$ and $b = 786$, except for one case on the SMP cluster. We also note that the speedup figures obtained on the SMP configuration were slightly better than the corresponding figures of the non-SMP configuration. Overall, two factors affect performance: the block factor b and the communication throughput $1/g$ of the PC cluster rather than latency related issues (e.g. L contributions).

From the expression for computation time in Theorem 1, nb must be kept smaller than $(n+1)^2/p$, or, in other words, the ratio b over $(n+1)/p$ must be minimized.

From the expression for communication time in Theorem 1 $2gn^2/p(b-1)$ must be kept smaller

than $(n+1)^2/p$ (sequential running time over p), or in other words the ratio $2g$ over b must be minimized. Similarly, the ratio $3L$ over nb/p must also be minimized.

For various processor sizes $p = 4 - 16$, the parameter L of the PC cluster varies for BSPlib between $900 - 1000\mu seconds$ to $3000 - 4000\mu seconds$. Since the effective megaflop rate of the application as derived from the execution time of the sequential implementation varies between $7.2Mflops$ ($n = 32768$) and $6.4Mflops$ ($n = 65536$), L varies between $6500-7000$ and $20000-25000$ operations (flops).

For the PC cluster of the experiments and the message sizes observed in the experiments, g varies between $10\mu seconds/double$ and $40\mu seconds/double$, values much higher than the asymptotic value of g which ranges between $0.64\mu seconds/double$ and $1.00\mu seconds/double$. The former values of g translated into operations per word (double), provide a range for g between 70 to 250 operations/word.

The range for L makes the ratio $3L/(nb/p)$ for $p = 16$, to vary from a high $0.03 - 0.06$ to $0.04 - 0.08$ for $n = 32768$ and $b = 512 - 1024$ to a low of $0.02 - 0.03$ and $0.02 - 0.04$ for $n = 65536$ and $b = 512 - 1024$. For such ranges of b and n algorithm BSP_Trinomial can be considered latency-tolerant since L-related contributions are negligible.

The parameter b however is much closer to $(n+1)/p$ for the given problem sizes. For $b = 640 - 786$, where the maximum speedup is observed, the ratio b over $(n+1)/p$ ranges from a high of $0.31 - 0.37$ for $n = 32768$ to a low of $0.15 - 0.19$ for $n = 65536$.

The ratio $2g/b$ affects performance more significantly. For $n = 32768$ and $b = 640 - 786$, the ratio ranges around $0.21 - 0.78$, and for $n = 65536$ and $b = 640 - 786$, it ranges around $0.17 - 0.63$.

Therefore for small problem sizes, communication, synchronization and the blocking structure of BSP_Trinomial in organizing the trinomial-tree valuation contribute a term that is as high as $51 - 123\%$ of the computation time for $n = 32768$, and as low as $34 - 100\%$ of the computation time for $n = 65536$. Therefore one should expect speedup ranges of $16/1.51 = 10.5$ to $16/2.23 = 7.1$ for $n = 32768$ and $16/1.34 = 11.9$ and $16/2.0 = 8$ for $n = 65536$.

The experimental results we obtained seem to be within the range of these rough estimates, as the speedup range obtained for $n = 32768$, and $p = 16$ is $7 - 8$ and for $n = 65536$ is $8.7 - 9.9$. This

supports the claim that the BSP model is an effective tool in parallel performance prediction.

We have also compiled the implementation of BSP_Trinomial and run similar experiments using LAM-MPI as the parallel communication library. The timing results obtained were similar to those obtained through *BSPlib* though worse for $p = 16, 32$ and slightly better in some cases for $p = 4$ and $p = 8$. The results obtained on the 16-node PC cluster are reported in Table 3 and the ones obtained in the SMP configuration (both CPUs of a single node were utilized) are reported in Table 4.

	$n = 32768$				$n = 65536$			
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
512	148.96	2.43	4.29	6.86	663.98	2.73	5.22	8.45
640	148.96	2.51	4.63	7.16	663.98	2.86	5.40	9.34
786	148.96	2.56	4.69	7.22	663.98	2.89	5.49	9.61
1024	148.96	2.54	4.66	6.99	663.98	2.89	5.53	9.53

Table 3: Execution Time and Speedup using Lam-MPI on a 16-node PC cluster.

	$n = 32768$					$n = 65536$				
	Time	Speedup				Time	Speedup			
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 32$
512	148.96	2.37	4.44	6.94	6.70	663.98	2.66	5.10	9.08	10.98
640	148.96	2.49	4.58	7.25	7.40	663.98	2.75	5.36	9.44	11.93
786	148.96	2.53	4.65	7.38	8.06	663.98	2.88	5.47	9.61	12.63
1024	148.96	2.51	4.62	7.44	8.37	663.98	2.83	5.54	9.81	13.37

Table 4: Execution Time and Speedup using Lam-MPI on an SMP cluster.

5 Conclusion

We have presented how parallel computations such as those involved in American-style option valuations can be performed in parallel in a trinomial-tree. We analyzed the parallel algorithm in an architecture independent setting using the tools provided by the BSP model of computation. We finally verified the practicality of our designs by implementing the proposed algorithm in not only an architecture independent but also library independent way and tested its performance in a cluster of PC workstations. The experimental results we obtained seem to confirm the conclusions drawn from the architecture independent analysis of the theoretical performance of the algorithm. The source code of the implementations is available through the author's web-page. This work was supported in part by NSF MRI grant NSF 9977508.

References

[1] G. Bilardi, K. T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis. BSP vs. LogP. In *Proceedings of the 8-th ACM Symposium on Parallel Algorithms and Architectures*, pp. 25-32, 1996.

[2] F. Black and M. Scholes. . The pricing of options and corporate liabilities. Journal of Political Economy, 81, 637-654.

[3] P.P.Boyle. . Option valuation using a three jump process. International Options Journal, 3, 7-12.

Implementing Derivative models. John Wiley and Sons, England, 1998.

[4] L. Clewlow and C. Strickland Implementing Derivative models. John Wiley and Sons, England, 1998.

[5] D. E. Culler and R. Karp and D. Patterson and A. Sahay and K. E. Schauser and E. Santos and R. Subramonian and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation.

Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Diego, CA, May 1993.

[6] A. C. Dusseau. Modeling parallel sorts with LogP on the CM-5. Technical Report UCB/CSD-94-829, Computer Science Division, University of California at Berkely, 1994.

[7] A. V. Gerbessiotis. `http://www.cs.njit.edu/~alexg/`, June 2002.

[8] A. V. Gerbessiotis and L. G. Valiant. Direct bulk-synchronous parallel algorithms. *Journal of Parallel and Distributed Computing,* 22:251-267, 1994.

[9] A. V. Gerbessiotis and C. J. Siniolakis. Ordered h-Level Graphs on the BSP Model. Journal of Parallel and Distributed Computing, 1998. *Journal of Parallel and Distributed Computing,* 49:98-110, Academic Press, 1998.

[10] A. V. Gerbessiotis and C. J. Siniolakis. Deterministic sorting and randomized median finding on the BSP model. In *Proceedings of the 8-th ACM Symposium on Parallel Algorithms and Architectures*, Padova, Italy, June 1996.

[11] A. V. Gerbessiotis and C. J. Siniolakis. Communication efficient data structures on the BSP model with applications in computational geometry. In *Proceedings of EURO-PAR'96*, Lyon, France, LNCS 1124, Springer-Verlag, August 1996.

[12] A. V. Gerbessiotis and C. J. Siniolakis. A randomized sorting algorithm on the BSP model. In Proceedings of IPPS, Geneva, Switzerland, IEEE Press, 1997.

[13] M.W. Goudreau, J.M.D. Hill, K. Lang, W.F. McColl, S.D. Rao, D.C. Stefanescu, T. Suel, and T. Tsantilas. A proposal for a BSP Worldwide standard. BSP Worldwide, `http://www.bsp-worldwide.org/`, April 1996.

[14] E.G.Haug The Complete guide to option pricing formulas. McGraw-Hill, 1998.

[15] D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and Answers about BSP. *Scientific Programming*, Vol 6, pp 249-274, 1997.

[16] LAM/MPI Parallel Computing. `http://www.lam-mpi.org`.

[17] E.S.Schwartz. The valuation of warrants: implementing a new approach. Journal of Financial Economics, 4, 79-94.

[18] W. F. Sharpe Investments Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[19] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM,* 33(8):103-111, August 1990.

[20] L. G. Valiant. General purpose parallel architectures. In *Handbook of Theoretical Computer Science,* (J. van Leeuwen, ed.), North Holland, 1990.