Architecture independent parallel binomial tree option price valuations

Alexandros V. Gerbessiotis

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

Research Report: CS-02-02

New Jersey Institute of Technology

Architecture independent parallel binomial tree option price valuations

Alexandros V. Gerbessiotis
CS Department
New Jersey Institute of Technology
Newark, NJ 07102
USA.

April 15, 2002

Abstract

We introduce an architecture independent approach in describing how computations such as those involved in American or European-style option valuations can be performed in parallel in the binomial-tree model. In particular we present an algorithm for the multiplicative binomial tree option-pricing model that can also be directly generalized to the general additive binomial tree model. The algorithm is described and analyzed in an architecture independent setting and running time and performance characteristics are expressed in terms of problem size n, which is the time horizon, and the parameters p, L and g of the bulk-synchronous parallel model of computation. In particular, our algorithm achieves optimal theoretical speedup $O(p)$ and is thus within a $1 + o(1)$ multiplicative factor of the corresponding sequential method. In order to verify the practicality of the introduced algorithm an experimental study of an implementation of the algorithm on a PC cluster is also presented; the implementation works with only recompilation of the source code under two popular programming libraries MPI (LAM-MPI) and BSPlib and is thus not only architecture but also communication library independent.

1 Introduction

Since the early days of computing it has been apparent that in the future sequential computers would be superseded by parallel machines. In machines built so far, however, communication is expensive, synchronization is not instantaneous and unlimited parallelism is not available. Clusters of PC or other workstations have been used in the past years as the platform of choice for running parallel or distributed applications. Given that in the past 25 years parallel algorithm design has been tilted towards shared memory, instantaneous synchronization and zero-latency models, many of the algorithms designed for such theoretical models do not perform well and are not very helpful in designing algorithms for the existing implementation platforms. In addition, the lack of theoretical tools and programming models that would allow the efficient design and analysis of parallel algorithms for the currently available platforms, has resulted in designing programs for such platforms in a trial-and-error way that significantly complicates the transportability and scalability of those programs. The introduction of realistic parallel computer models such as the Bulk-Synchronous Parallel (BSP) model [15, 16] and the LogP [3] model aim at addressing these limitations of parallel computing.

In this work, we deal with the binomial tree valuations of assets such as American-style stock options. We propose a new way in dealing with these valuations in parallel using space proportional to the time horizon of the valuation and performing the valuation related computations in parallel time that is optimal, thus achieving optimal parallel speedup.

The computational model adopted is the Bulk-Synchronous Parallel (BSP) model of computation, which deals explicitly with the notion of communication and synchronization among computational threads. The *Bulk-Synchronous Parallel* (BSP) model of computation has been proposed by L.G. Valiant [15, 16] 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.

(1) 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.

(2) A *communication network* that can deliver messages point to point among the components.

(3) 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.

(1) p - the number of processor components .

(2) L - the minimum elapsed time, measured in basic computational operations, between successive synchronization operations.

(3) 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 $h = \max\{h_t, h_r\}$.

1.1 Specifying the Performance of BSP Algorithms

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 Binomial tree valuations

The binomial model was introduced in [14] for pricing primarily American-style options [2]. The binomial valuation model [14] assumes that the underlying asset price follows a binomial process, so that at any time the asset price can only change into one of two possible values. In an equal-jump multiplicative binomial process, for an asset whose current price is S, during time Δt, the asset price can go up to uS or down to dS. Parameters u and d describe the average behavior and also the volatility of the considered asset. A call option (intend to buy at a predetermined strike price) with strike price K which matures at Δt time is worth $\max\{0, uS - K\}$ if the price goes up and $\max\{0, dS - K\}$ otherwise. If an investor has a long position of Δ shares of the stock, and also has a short position of one call option, the investor would like that the portfolio remain the same regardless of the valuation of the asset price at Δt, in other words

$$-\max\{0, uS - K\} + \Delta uS = -\max\{0, dS - K\} + \Delta dS,$$

which results in

$$\Delta = \frac{\max\{0, uS - K\} - \max\{0, dS - K\}}{(u - d)S}.$$

Within time Δt the portfolio should earn the riskless and continuously compounded interest rate r, i.e.

$$-\max\{0, uS - K\} + \Delta uS = e^{r\Delta t}(-C + \Delta S),$$

where C is the fair value of the option at the start of the period. By defining $p = (e^{r\Delta t} - d)/(u - d)$ and solving for C, we derive the option price at the start of the period. In order to price options with more that one time period to maturity we extend the binomial tree for the number of required time periods to maturity. In general for a binomial tree with n time periods to maturity we would label nodes of the tree with the pair (i, j), where i indicates the number of time steps from the start (time zero) and j indicates the number of downward movements the asset price has made since start. Therefore the asset price at (i, j) is $S_{i,j} = Sd^j u^{i-j}$ and the option price is assumed to be $C_{i,j}$. The value of the option at maturity is the payoff i.e.

$$C_{n,j} = \max\{0, S_{n,j} - K\}. \tag{1}$$

Figure 1. shows a binomial tree with time horizon $n = 8$. A node (i, j) is connected to node $(i, j + 1)$ through a horizontal link and to node $(i + 1, j + 1)$ through a diagonal link.

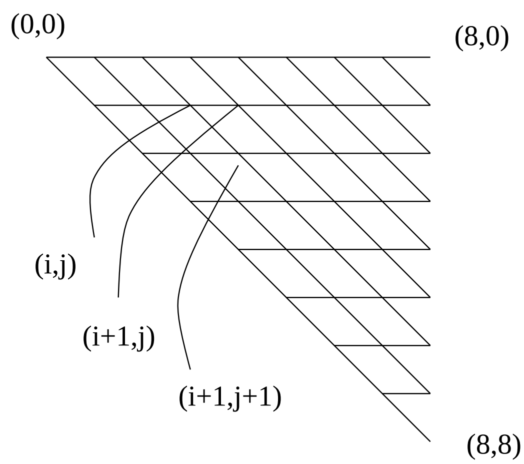

Figure 1: A Binomial Tree with n=8.

For an American call option (that can be exercised at any time before or on maturity),

$$C_{i,j} = \max\{e^{-r\Delta t}(pC_{i+1,j} + (1 - p)C_{i+1,j+1}), S_{i,j} - K\} \tag{2}$$

whereas for a European call option (that can be exercised on maturity only)

$$C_{i,j} = e^{-r\Delta t}(pC_{i+1,j} + (1 - p)C_{i+1,j+1}) \tag{3}$$

Eq. (1) along with Eq. (2) or Eq. (3) can be used to compute the option price at every time period and node of the binomial tree. The option price at the start of the period is thus $C_{0,0}$.

Figure 2 provides pseudocode for European call option pricing on an equal jump multiplicative binomial tree.

```
begin EUROPEAN_CALL (K,n, Δt, S, r, u, d)
    let p = (e^{rΔt} − d)/(u − d) ;
    let discount = e^{−rΔt} ;
    let S[0] = S ∗ u^n ;
    for j = 1 to n do
        let S[j] = S[j − 1] ∗ (d/u);
    for j = 0 to n do
        let C[j] = max{0, S[j] − K} ;
    for i = n − 1 downto 0 do
        for j = 0 to i do
            let C[j] = discount ∗ (p ∗ C[j] + (1 − p) ∗ C[j + 1]) ;
    return (C[0]);
end EUROPEAN_CALL
```

Figure 2: Procedure EUROPEAN_CALL.

For an American-style call option the equivalent pseudocode is presented in Figure 3.

We finally note that the introduced algorithm can be easily generalized to the general additive binomial-tree model [2].

3 Parallel Multiplicative Binomial Tree Valuations

For a valuation with horizon n, the binomial tree has a total of $n(n+1)/2$ 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. 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 need to be allocated. The pseudocode in Figure (2) and Figure (3) reflect this assumption. If on the other hand all intermediate results need to be maintained, the total space requirements become quadratic in n. We examine the former case in this work, although the results described hold for the other case as well, as time requirements are the same for both cases.

```
begin AMERICAN_CALL (K, n, Δt, S, r, u, d)
    let p = (e^{rΔt} − d)/(u − d) ;
    let discount = e^{−rΔt} ;
    let S[0] = S ∗ u^n ;
    for j = 1 to n do
        let S[j] = S[j − 1] ∗ (d/u);
    for j = 0 to n do
        let C[j] = max{0, S[j] − K} ;
    for i = n − 1 downto 0 do
        for j = 0 to i do
            let C[j] = discount ∗ (p ∗ C[j] + (1 − p) ∗ C[j + 1]) ;
            let S[j] = S[j]/u ; // one less up movement
            let C[j] = max{C[j], S[j] − K};
    return (C[0]);
end AMERICAN_CALL
```

Figure 3: Procedure AMERICAN_CALL.

Information related to option price is maintained in array C and stock price in array S by allocating space proportional to $n+1$, 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 elements to processor 1, …, the next b elements to processor $p-1$, and wrapping around for the first time, the next b elements to processor 0, and so on.

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 $b = \omega(1)$.

For the sake of an example, Figure 4 shows the top 8 levels of a binomial tree with horizon $n \geq 8$. Only selected links of a subset of the nodes of the binomial tree are shown. The chosen value of b is $b = 4$. The j indices of the nodes of the rightmost level of the binomial tree fragment

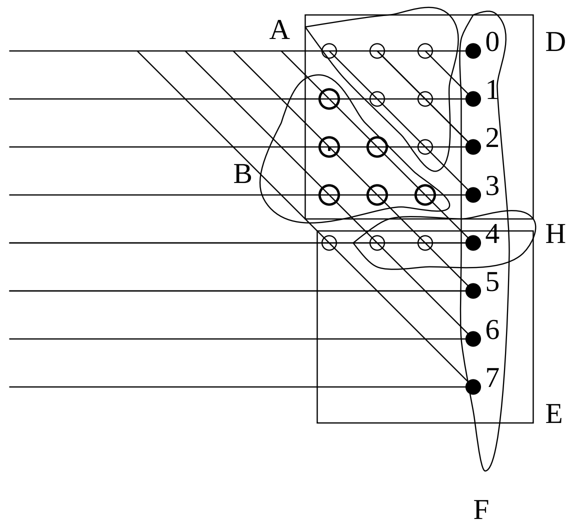

Figure 4: Binomial Tree calculations.

are also shown.

In a parallel valuation of the operations performed on a binomial tree, the introduced parallel algorithm walks through the time periods from n to 0, by processing b periods per computation/communication round. This is to avoid frequent communication and synchronization operations. Given the memory allocation scheme described in the introduction of this section, processors thus operate on square $b \times b$ blocks of the tree, and each processor maintains in its memory multiple such blocks (except perhaps for the last few time periods).

With reference to Figure 4, for the chosen block factor $b = 4$ the first blocks assigned to processors 0 and 1 are the sets of nodes D and E respectively (sketched square boxes). The nodes of set F are the ones initialized by way of lines 5 and 7 of Figure 2 (i.e. C values have already been assigned by way of line 7). By tracing the computations of pseudocode of Figure 2 performed on the nodes of set D we conclude that the nodes of set A can be valuated by using information locally available to processor 0 by way of the loop of lines 8 and 9. This is not possible, however, for the nodes of set B as they require information available in processor 1. This information is maintained by the set H of nodes in processor 1. The same situation occurs with every other adjacent pair of processors that participates in this round of valuation. Every processor m thus needs to communicate the information available by its own set H of nodes to processor $m - 1$, and then and only then, can processor $m - 1$ perform the computations of lines 8 and 9 on its own set B of nodes by utilizing the received information related to the H nodes.

Therefore a parallel valuation of a binomial tree valuates b levels of the tree at a time; in each such valuation a computation superstep (computation on the set A of nodes) is followed by a communication superstep (communication of C values of the set H of nodes), and concluded by a computation superstep (computation on the set B of nodes). The following pseudocode for BSP_MBT describes the major steps of a parallel binomial tree valuation.

begin BSP_MBT ()
1. Initialize level n; //i.e. assign $C_{n,j}$ values to nodes of set F
2. **for** ($k = n; k > 0; k = k - b + 1$)
3. Perform computation of loop 8 and 9 for nodes of set A;
4. Communicate $C_{i,j}$ values for nodes of set H from every processor m to processor $m - 1$;
5. Perform computation of loop 8 and 9 for nodes of set B using the information received in the previous step;
6. //Prices C_{ij} for time $k - b + 1, \ldots, k$ have been valuated.

end BSP_MBT

Figure 5: Procedure BINOMIAL_TREE_VALUATION.

3.1 Time analysis of the algorithm

With reference to Figure 5 for algorithm BSP_MBT, we describe the space and time requirements of the parallelization of the valuation method described in Figure 3 or Figure 2; all references to pseudocode in Figure 5 refer to the pseudocode of Figure 3 or Figure 2. Note that C become a two-dimensional array in Figure 5 to facilitate exposition only.

Line 1. The $n + 1$ nodes of the binomial tree at time n are split into $\lceil (n + 1)/b \rceil$ groups of size b, and each processor gets at most $\lceil \lceil (n + 1)/b \rceil /p \rceil$ groups. Each group consists of at most b keys. The initialization step of line 1 of Figure 5 thus requires time proportional to $\lceil \lceil (n + 1)/b \rceil /p \rceil \leq ((n + 1)/(pb) + 1)b = (n + 1)/p + b$.

Loop of line 2. The loop of line 2 starts at $k = n$, and every time it is executed, all the nodes of the tree of time periods $k - b + 1 \ldots k - 1$ are valuated. We call the valuation of all the nodes of the binomial tree in a single iteration of line 2, an update. The first update takes place for $k = n$, the second for $k = n - (b - 1)$, and similarly the i-th update is given as a function of n, k and b by the following expression: $i = (n - k + (b - 1))/(b - 1)$. Later on references to time periods involve references to k. Multiple time periods are involved in a single update; a reference to an update will involve references to i. 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 line 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 $b-1$.

Lines 4 and 6. Each processor is assigned a number of square $b \times b$ blocks of the binomial tree, except perhaps for the last block of the last processor that also maintains a block whose active nodes (that need to be valuated) is an upper diagonal array/block. In the update that includes the k-th time period (i.e. the i-th update) the number of nodes of the binomial tree at that time period k is $k+1$ and therefore each one of the p processors will valuate at most $\lceil \lceil (n+1-i(b-1))/b \rceil /p \rceil$ blocks of dimension $b \times b$. On each such block $b(b-1)$ node updates need to be performed i.e. $C_{i,j}$ values will be assigned to the nodes of sets A and B of a set like set D in Figure 4. The total work required for the i-th update is thus equal to $\lceil \lceil (n+1-i(b-1))/b \rceil /p \rceil b(b-1)$ 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 remaining calculations below. The parallel computation time of lines 4 and 6 is thus given by the following expression.

$$\begin{aligned}
\sum_{i=1}^{\lceil \frac{n}{(b-1)} \rceil} \left(\left\lceil \frac{\left\lceil \frac{n+1-i(b-1)}{b} \right\rceil}{p} \right\rceil b(b-1) \right) &= \sum_{i=1}^{\lceil \frac{n}{(b-1)} \rceil} \left(\frac{n+1-i(b-1)}{bp} + 1 \right) b(b-1) \\
&= \sum_{i=1}^{\lceil \frac{n}{(b-1)} \rceil} \left(\frac{(n+1)(b-1)}{p} - \frac{i(b-1)^2}{p} + b(b-1) \right) \\
&= \frac{(n+1)(b-1)}{p} \left\lceil \frac{n}{b-1} \right\rceil + b(b-1) \left\lceil \frac{n}{b-1} \right\rceil - \\
&\quad \frac{(b-1)^2}{2p} \left\lceil \frac{n}{b-1} \right\rceil \left(\left\lceil \frac{n}{b-1} \right\rceil + 1 \right) \\
&= \frac{n^2+2n}{2p} + nb - \frac{n(b-1)}{2p}. \qquad (4)
\end{aligned}$$

Line 5. The communication performed in line 5 requires the communication of $(b-1)$ $C_{i,j}$ values per $b \times b$ block. Therefore the communication and synchronization time is bounded above by the following expression. The upper bound at the very end of the derivations is derived provided that $b \geq 3$.

$$
\begin{aligned}
\sum_{i=1}^{\left\lceil \frac{n}{(b-1)} \right\rceil} \max\left\{L, \left\lceil \frac{\left\lceil \frac{n+1-i(b-1)}{b} \right\rceil}{p} \right\rceil (b-1)g \right\} &= \sum_{i=1}^{\left\lceil \frac{n}{(b-1)} \right\rceil} \max\left\{L, \left(\frac{n+1-i(b-1)}{bp}+1\right)(b-1)g\right\} \\
&= L\left\lceil \frac{n}{(b-1)} \right\rceil + \\
&\quad \sum_{i=1}^{\left\lceil \frac{n}{(b-1)} \right\rceil} g \cdot \left(\frac{(n+1)(b-1)}{pb} - \frac{i(b-1)^2}{pb} + (b-1)\right) \\
&= L\left\lceil \frac{n}{(b-1)} \right\rceil + g\frac{(n+1)(b-1)}{pb}\left\lceil \frac{n}{(b-1)} \right\rceil + \\
&\quad g(b-1)\left\lceil \frac{n}{(b-1)} \right\rceil + \\
&\quad -g\frac{(b-1)^2}{pb}\left\lceil \frac{n}{(b-1)} \right\rceil \left(\left\lceil \frac{n}{b-1} \right\rceil + 1\right)/2 \\
&= L\frac{n}{(b-1)} + g\frac{n(n+1)}{pb} + gn - g\frac{n^2}{2pb} - g\frac{(b-1)n}{2pb} \\
&= L\frac{n}{(b-1)} + g\frac{n^2}{2pb} + gn - g\frac{(b-3)n}{2pb} \\
&\leq L\frac{n}{(b-1)} + g\frac{n^2}{2pb} + gn. \qquad (5)
\end{aligned}
$$

The total parallel computation time results by combining the time for line 1 with that for lines 4 and 6. Therefore we get the following expression.

$$
\begin{aligned}
C(n,p) &= \frac{n^2+2n}{2p} + nb - \frac{n(b-1)}{2p} + \frac{n+1}{p} + b \\
&= \frac{(n+1)(n+2)}{2p} + \frac{n}{2p} + (n+1)b - \frac{n(b-1)}{2p} \\
&\leq \frac{(n+1)(n+2)}{2p} + \frac{n}{2p} + (n+1)b
\end{aligned}
\qquad (6)
$$

We note that the sequential computation time is given by $T(n) = \sum_{i=1}^{n+1} i = (n+1)(n+2)/2$.

The total parallel communication time is the one given by Eq. (5). 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_MBT *for the valuation of an equal jump multiplicative binomial tree requires parallel computation time equal to*

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

and parallel communication (and synchronization) time

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

One optimality for BSP_MBT 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, $p \leq n$, $b \geq 3$ such that $b-1$ divides n, algorithm* BSP_MBT *for $b = o(n/p)$, $L = o(nb/p)$, and $g = o(b)$ achieves $\pi = 1 + o(1)$ and $\mu = o(1)$.*

Proof. For computation time, by Theorem 1 we have

$$C(n,p) \leq \frac{(n+1)(n+2)}{2p} + \frac{n}{2p} + (n+1)b.$$

Similarly, for communication time by the same theorem we derive for $b \geq 3$ that

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

For the BSP parameter L as long as $Ln/(b-1) = o(n^2/p)$ 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, blocking is reflected in the extra 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. ■

We note that the results described for an equal jump multiplicative binomial tree model can be extended to the general additive binomial tree model as well.

4 Performance Evaluation

As the BSP model is not just an abstract architectural model, it may also be employed as a programming platform or, indeed, as a kind of a programming paradigm. The underlying concept in the BSP model is the notion of the superstep and the abstraction that non-local communication associated with a superstep takes place between supersteps as a global operation. Thus, a BSP program may be viewed as a succession of supersteps, with the required non-local communication occurring at the end of each superstep. Viewed this way, the BSP model can 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 [12] 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. One needs to understand the semantics of no more than 10-15 functions; half of these functions are related to process creation, destruction and identification.

It should be borne in mind however, that the library support such as that offered by a BSP model-specific library such as the Oxford BSP Toolset 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. We have thus made our BSP code that implements BSP_MBT 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 [13] 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 implementations presented in this work were programmed in ANSI C. Our implementations are test for scalability and portability on a cluster of PC workstations consisting of 16 dual-Pentium 350Mhz Redhat Linux 7.1 workstations, with 128MB of memory, and equipped with 100Mbit 3Com-905B Ethernet cards. The PC workstations are all connected on 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* [11] 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 as well as its extension to the general additive binomial tree model has been made publically available at the author's Web-page [5]. Both programs can be compiled and run under either *BSPlib* or LAM-MPI.

The timing results obtained are discussed later in this section. The depicted results in the following tables are in general, averages over at least four experiments performed in close succession. The tested platform is either the 16-node PC cluster utilizing only one CPU per node, or an 8-node subcluster utilizing both CPUs. By setting up the cluster this way we wanted to examine whether the proximity of SMP processor can improve performance. The reported speedup results reflect valuations of American-style put options. We then report on the performance and relative merits of the implementation of BSP_MBT. Timing and/or speedup figures are in general truncated to two or fewer decimal digits. We performed experiments for various problem sizes $n = 8192, 16384, 32768, 65536$. We also varied block size b; in the experiments we set $b = 32, 64, 128, 256, 384$. For $p = 1$ timing results are reported, which are independent of b. American-style put options are valuated by a straightforward algorithm similar to the one depicted in Figure 3; we thus did not run BSP_MBT on a single processor to obtain these timing results. For other processor sizes $p = 4, 8, 16$, relative speedup figures are only reported rather than actual timing results; it is straightforward to obtain time from speedup figures and the $p = 1$ timing results.

Table 1 shows timing results for problem sizes $n = 8192$ and $n = 16384$, and Table 2 for problem sizes $n = 32768$ and $n = 65536$ on the 16-node PC cluster. Table 3 shows timing results for problem sizes $n = 8192$ and $n = 16384$, and Table 4 for problem sizes $n = 32768$ and $n = 65536$ on an 8-node (16 processor) subcluster.

Table 1 and Table 2 indicate the increasing performance of BSP_MBT for increasing problem size. The best performance of the algorithm is obtained when block factor b is between $b = 64$ and

		$n = 8192$				$n = 16384$		
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	4.60	1.52	2.54	2.97	30.65	2.26	4.17	5.28
64	4.60	1.69	2.86	3.80	30.65	2.49	4.59	6.68
128	4.60	1.68	2.71	3.19	30.65	2.53	4.47	6.56
256	4.60	1.53	1.99	2.36	30.65	2.39	3.98	5.07
384	4.60	1.32	1.66	1.90	30.65	2.25	3.38	3.93

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

		$n = 32768$				$n = 65536$		
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	155.30	2.58	4.92	5.07	753.66	2.83	5.38	5.77
64	155.30	2.80	5.27	7.73	753.66	3.09	5.81	8.98
128	155.30	2.88	5.43	8.85	753.66	3.21	6.23	10.88
256	155.30	2.83	5.11	8.00	753.66	3.21	6.11	10.64
384	155.30	2.76	4.74	7.26	753.66	3.18	5.91	9.99

Table 2: Execution Time and Speedup using BSPlib on a 16-node PC cluster.

$b = 128$ for the problem sizes in Table 1 and $b = 128$ and $b = 256$ for the problem sizes in Table 2. The same observations apply to Table 3 and Table 4. We also note that the speedup figures obtained on 16 processors set-up as an 8-node 2-cpu-per-node subcluster were consistently better than those obtained on 16 processor set-up as a 16-node 1-cpu-per-node subcluster. Overall, two factors affect performance: the block factor b and the communication throughput $1/g$ of the PC cluster.

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

From the expression for communication time in Theorem 1 $gn^2/2pb$ must be kept smaller than $n^2/2p$ (sequential running time), or in other words the ratio g over b must be minimized. Similarly, the ratio L over nb/p must also be minimized.

For various processor sizes $p = 4 - 16$, the parameter L of the PC cluster varies for BSPlib

	$n = 8192$				$n = 16384$			
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	4.60	1.53	2.50	3.15	30.65	2.29	6.15	5.35
64	4.60	1.71	2.86	3.87	30.65	2.50	4.57	7.11
128	4.60	1.70	2.71	3.15	30.65	2.55	4.51	6.59
256	4.60	1.51	2.01	2.36	30.65	2.42	4.01	5.25
384	4.60	1.35	1.68	1.90	30.65	2.29	3.40	4.01

Table 3: Execution Time and Speedup using BSPlib on an 8-node SMP cluster.

	$n = 32768$				$n = 65536$			
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	155.30	2.60	4.96	6.34	753.66	2.85	5.39	8.04
64	155.30	2.81	5.34	7.72	753.66	3.09	5.68	8.79
128	155.30	2.91	5.45	9.12	753.66	3.22	6.26	11.30
256	155.30	2.86	5.13	8.27	753.66	3.22	6.13	10.98
384	155.30	2.79	4.76	7.42	753.66	3.20	5.93	10.19

Table 4: Execution Time and Speedup using BSPlib on an 8-node SMP cluster.

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 $7Mflops$ ($n = 8192$) and $3Mflops$ ($n = 65536$), L varies between 6000 and 12000 operations.

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 120 operations/word.

The range for L of $6000 - 12000$ makes the ratio $L/(nb/p)$ for $p = 16$, to vary from a high of $0.2 - 0.36$ for $n = 8192$ and $b = 64 - 128$ to a low of $0.01 - 0.03$ for $n = 65536$ and $b = 128 - 256$. For such ranges of b and n algorithm BSP_MBT can be considered latency-tolerant since L-related contributions are small.

The parameter b however is much closer to $n/2p$ for the given problem sizes. For $b = 64 - 256$, where the maximum speedup is observed, the ratio b over $n/2p$ ranges from a high of $0.25 - 0.5$ for $n = 8192$ and $b = 64 - 128$ to a low of $0.07 - 0.125$ for $n = 65536$ and $b = 128 - 256$.

The ratio g/b affects performance more significantly. For $n = 8192$ and $b = 64 - 128$, the ratio ranges around $0.60 - 1.1$, and for $n = 65536$ and $b = 128 - 256$, it ranges around $0.5 - 1$.

Therefore for small problem sizes, communication, synchronization and the blocking structure of BSP_MBT in organizing the binomial-tree valuation contribute a term that is as high as $105-200\%$ of the computation time for $n = 8192$, and as low as between $58-100\%$ of the computation time for $n = 65536$. Therefore one should expect speedup ranges of $16/3 = 5.3$ to $16/2.0 = 8$ for $n = 8192$ and $16/2.0 = 8$ and $16/1.58 \approx 10$ for $n = 65536$.

The experimental results we obtained seem to be within the range of these rough estimates, as the maximum speedup obtained for $n = 8192$, and $p = 16$ is no more than 3.8 and for $n = 65536$ is approximately 10.88. This supports the claim that the BSP model is an effective tool in parallel performance prediction.

In addition to algorithm BSP_MBT we have also implemented algorithm BSP_GBT for parallel option price valuations for the general additive binomial tree. The results we obtained are similar to the ones reported here except that for $p = 16$ speedup results were better by at least 5% in few cases to as much as 10% in most cases.

We have also compiled the implementation of BSP_MBT and run similar experiments using LAM-MPI as the parallel communication library. The timing results obtained were similar to those obtained through *BSPlib* though a little (and consistently) worse. The results obtained on the 16-node PC cluster are reported in Table 5 and Table 6 and the ones obtained in the 8-node PC cluster (both CPUs were utilized) are reported in Table 7 and Table 8.

5 Conclusion

We have presented an architecture independent approach in describing how computations such as those involved in American or European-style option valuations can be performed in parallel in a binomial tree. We analyzed the 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

		$n = 8192$				$n = 16384$		
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	4.60	0.62	0.55	0.83	30.65	1.01	1.63	1.93
64	4.60	1.19	1.72	1.68	30.65	1.84	3.08	3.83
128	4.60	1.50	2.23	2.28	30.65	2.32	3.93	5.26
256	4.60	1.45	1.87	1.97	30.65	2.34	3.77	4.71
384	4.60	1.30	1.60	1.74	30.65	2.22	3.30	3.69

Table 5: Execution Time and Speedup using Lam-MPI on a 16-node PC cluster.

		$n = 32768$				$n = 65536$		
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	155.30	1.22	2.08	2.93	753.66	1.41	2.42	3.65
64	155.30	2.14	3.82	5.75	753.66	2.38	4.51	7.44
128	155.30	2.69	4.91	7.74	753.66	3.02	5.71	10.03
256	155.30	2.78	4.93	7.69	753.66	3.18	5.95	10.45
384	155.30	2.73	4.66	7.12	753.66	3.16	5.82	9.94

Table 6: Execution Time and Speedup using Lam-MPI on a 16-node PC cluster.

we obtained seem to confirm the conclusions drawn from the architecture independent analysis of the theoretical performance of the algorithm.

6 Acknowledgements

This work was supported in part by NJ SBR grant 421350 and 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.

		$n = 8192$				$n = 16384$		
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	4.60	0.67	0.99	0.88	30.65	1.09	1.76	1.99
64	4.60	1.26	1.83	1.72	30.65	1.93	3.22	3.96
128	4.60	1.59	2.23	2.29	30.65	2.40	4.02	5.30
256	4.60	1.48	1.91	2.05	30.65	2.39	3.85	4.74
384	4.60	1.35	1.65	1.78	30.65	2.29	3.34	3.77

Table 7: Execution Time and Speedup using Lam-MPI on an 8-node SMP cluster.

		$n = 32768$				$n = 65536$		
	Time	Speedup			Time	Speedup		
b	$p = 1$	$p = 4$	$p = 8$	$p = 16$	$p = 1$	$p = 4$	$p = 8$	$p = 16$
32	155.30	1.33	2.25	3.11	753.66	1.54	2.60	3.74
64	155.30	2.25	3.96	5.89	753.66	2.50	4.64	7.60
128	155.30	2.75	4.98	7.79	753.66	3.08	5.80	10.06
256	155.30	2.83	4.99	7.62	753.66	3.19	5.98	10.45
384	155.30	2.78	4.68	7.13	753.66	3.21	5.85	9.94

Table 8: Execution Time and Speedup using Lam-MPI on an 8-node SMP cluster.

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

[3] 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.

[4] 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.

[5] A. V. Gerbessiotis. `http://www.cs.njit.edu/~alexg/`, April 1996.

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

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

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

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

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