Fast parallel solution of fixed point equations for the performance evaluation of circuit-switched networks

Albert G. Greenberg[a], Andrew M. Odlyzko[a], Jennifer Rexford[b*], and David Espinosa[c†]

[a]Mathematical Sciences Research, AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

[b]Dept. of Computer Science and Engineering, Real-Time Computing Laboratory, Room 2220, EECS Building, 1301 Beal Avenue, Ann Arbor, MI 48109-2122

[c]Computer Science Dept., Columbia University, 450 Computer Science, New York, N.Y. 10027

Massively parallel algorithms are presented for solving systems of fixed point equations, modeling state-dependent routing in large asymmetric circuit-switched networks. Our focus is on the Aggregated Least Busy Alternative (ALBA) routing policy of Mitra, Gibbens and Huang. On a 16384 processor MasPar parallel computer, about a minute is required to compute estimates of the call blocking probabilities for every node-pair, for realistic networks of over 100 nodes. A few hours are required on a high speed workstation.

1. Introduction

The engineering of a modern telecommunication system involves extensive performance evaluation, testing the system's response to a gamut of stresses, including traffic overloads and equipment failures such as fiber cuts. At present, the dominant tool for such performance evaluation is discrete event simulation. In this paper, we consider an alternative: analytic fixed point methods, which yield approximate performance estimates. Massively parallel algorithms are presented for solving the equations. The algorithms have been implemented and tested on a 16384 processor MasPar parallel computer. Experimental results, for realistic simulation scenarios for large asymmetric networks, show that accurate estimates are delivered at great speed. Thus, the algorithms are fast enough to provide an interactive means for exploring network performance, which should serve as a useful complement to discrete event simulation.

We consider networks of N nodes in which a stream of calls arrives to each node pair, with each call representing a request that a route in the network be set up to carry the call. It is convenient to assume that the network is fully connected; there is no loss of generality because we allow a link's call carrying capacity, counted in trunks, to be 0. If accepted, a call arriving to the stream associated with a given node pair (i, j) requires for its exclusive use one trunk on each link of its route between i and j.

*Jennifer Rexford's research was supported by a grant from the AT&T Graduate Research Program for Women.

†David Espinosa's research was supported by an AT&T graduate fellowship.

Mitra, Gibbens, and Huang [14] introduced the Aggregated Least Busy Alternative (ALBA) state dependent routing scheme. This policy is simple and remarkably effective, and is closely related to the Real-Time-Network-Routing (RTNR) policy, used to control the AT&T long distance network [2]. The number of calls in progress on each link determines the link's *aggregate* state, in the range 0 to K, where K is a small parameter (e.g., $K = 2$ or 3). A low aggregate state indicates relatively few calls in progress. Alternate routes are restricted to two links: $\{(i,k),(k,j)\}$, for any $k \neq i,j$. A link (i,j) in state $\geq K-1$ is *reserved* for the use of stream (i,j), and cannot be used as part of any alternate route. A call arriving to stream (i,j) is routed on link (i,j) unless that link is full to capacity. Otherwise, if at least one alternate route is not reserved, then an alternate route is chosen uniformly at random among those in the least aggregate state. Otherwise, the call is blocked (rejected and lost). The details are described in Section 2.

Following the treatment in [14] for symmetric networks, we derive a system of "fixed point" equations that estimate traffic flows under ALBA for asymmetric networks. Unfortunately, for large, realistic networks, the system of equations is so large and computationally burdensome to solve on a conventional, serial computer, that solving the equations appears to offer little advantage over discrete event simulation.

Our contribution is to show that the parallelism inherent in the system of fixed point equations can be simply and efficiently exploited to solve the system very rapidly on today's massively parallel computers. Our algorithms have been implemented on a 16384 processor MasPar [4] single-instruction-multiple-data (SIMD) parallel computer, and have been found to be numerically robust as well as fast. It is a simple matter to adapt the codes to a large class of state-dependent routing schemes other than ALBA (Section 6).

In Section 2, we define the network model and the ALBA routing policy. In Section 3, we describe the system of fixed point equations, and in the following section present our algorithms for the solution. A natural parallel architecture for the computational problem is the $N \times N$ mesh, described in Section 4. The solution method is iterative. On the $N \times N$ mesh, each iteration takes $O(KN + C_{max})$ time, where C_{max} is the maximum link capacity. On a serial computer the time needed is $O(KN^3 + C_{max}N^2)$. In Section 5, we report our computational experience. Some final remarks are given in Section 6.

In recent years, there has been considerable research in the area of analysis of state-dependent routing policies through the solution of fixed point equations [1, 3, 6–11, 13, 15, 16]. Independently of the work reported here, Chung, Kashper, and Ross [5] have developed, implemented, and tested serial algorithms for solving the fixed point equations for networks with state-dependent routing. Their results include an interesting time/memory tradeoff, and a new approximation for multirate networks.

Our approach to the derivation of the equations, which closely follows the development in [14] is different. In particular, our equations model the uniform random choice of an alternate two-link paths among all such paths in the least aggregate state. Chung, Kashper, and Ross [5] investigate "least loaded routing" (an extreme case of ALBA, where the number of aggregate classes K is unbounded) in detail, and assume a deterministic tie-breaking rule rather than uniform random choice. Under least loaded routing, the tie-breaking rule may have little impact because ties may be infrequent. In the case of ALBA, there are compelling performance and practical reasons for very small K [14]. Nearly all alternate routing opportunities would require tie-breaking, and it makes sense to spread the overflow traffic uniformly over the alternates in the least aggregate state. Indeed, this is done in pseudo-random fashion in the AT&T RTNR scheme. Handling uniform random choice turns out to be significantly more computationally challenging than deterministic tie-breaking. In Section 3.1, we propose solutions based on an integral representation of the choice probabilities.

Chung, Kashper, and Ross [5] assess the accuracy of the estimates under a variety of traffic loads. Generally, their experiments show that the accuracy is quite acceptable, and is especially good at moderate and heavy loads. However, at light loads, where blocking is rare, the blocking estimates are too optimistic–predicting that blocking is rarer still. (Our experiments–not reported here–confirm these trends.) In Section 5 we test our algorithm on a realistic focused overload scenario with parameters modeled after the AT&T network. (It turns out that the overall call blocking probability is about 6%, and congestion is concentrated on several hundred of the several thousand links.) It is of critical importance to investigate network performance and reliability under such scenarios. We find in Section 5 the accuracy of the blocking estimates to be quite acceptable; showing remarkable agreement with simulation.

2. Network Model

We consider a fully connected, circuit-switched network with N nodes and $N(N-1)/2$ bidirectional links. It is convenient to use the tuple (i,j) to denote the pair of nodes i and j, keeping in mind that (j,i) denotes the same pair; $0 \leq i,j < N$. The basic parameters of node-pair (i,j) are

- the capacity of link (i,j), counted in trunks, and
- the rate of arrival of the stream of calls to (i,j).

We write these two parameters as functions of (i,j): $C(i,j)$ and $\lambda(i,j)$, respectively. Again, since (i,j) and (j,i) denote the same node-pair, these and all functions of (i,j) introduced below have the same value at (j,i) as at (i,j). It is assumed that call arrival process to each stream is Poisson, and that each call's duration is independently and exponentially distributed with mean 1. (Thus, rates are counted in erlangs.)

Aggregated Least Busy Alternative

A call arriving to stream (i,j) is routed on the direct one-link path, if at the time of arrival, at least one trunk on the link is free. Otherwise, the call is either routed on some two-link path between i and j, or blocked. At the time of its arrival, provided the call is not blocked, the call seizes for its exclusive use one trunk on each link of its route, and releases those trunks at the time of its departure. We term a call routed on a one-link path a *direct call* and one routed on a two-link path an *overflow call*.

Under the Aggregated Least Busy Alternative (ALBA) routing policy [14], the choice of the two-link path for an overflow call depends on the instantaneous network state, and favors paths whose current load is relatively light. Consider link (i,j), with capacity $C = C(i,j)$, and *state* defined as the number of calls in progress on the link. A key control parameter is the number of *aggregate states* $K+1$ the link may assume. Aggregate states

$\mathcal{A}_I = \mathcal{A}_I(i,j)$, $I = 0, ..., K$, are determined by parameters $r_J = r_J(i,j)$, $J = 1, ..., K-1$, as

$$\begin{aligned}
\mathcal{A}_0 &= \{0, 1, \ldots, C - r_1 - 1\}, \\
\mathcal{A}_1 &= \{C - r_1, C - r_1 + 1, \ldots, C - r_2 - 1\}, \\
&\vdots \\
\mathcal{A}_{K-1} &= \{C - r_{K-1}, C - r_{K-1} + 1, \ldots, C - 1\}, \\
\mathcal{A}_K &= \{C\},
\end{aligned}$$

An aggregate $\mathcal{A}_I$ may be empty. Letting n denote the link's state, the aggregate state is that I such that $n \epsilon \mathcal{A}_I$. (These definitions of sets $\mathcal{A}_I$ differ slightly from the definitions in [14].)

Alternate routing works as follows. The aggregate state of a two-link alternate path is defined to be the maximum of the aggregate states of the two links. If there is at least one alternate path with aggregate state less than $K-1$, then the call is carried on one of the paths, selected uniformly at random from among those having the minimal aggregate state. Otherwise, the call is blocked. Thus, links with aggregate state $K-1$ are available only for direct calls, while links in lower aggregate states are available for direct and overflow calls. In aggregate state K, the link is full to capacity, so no calls can be accepted.

3. Fixed Point Equations

In this section, we present the system of fixed point equations describing the network's equilibrium behavior, under the following independence approximation: *The event that a given link assumes a given aggregate state is independent from link to link.* The unknowns in the system of equations are the state dependent overflow rates of calls impinging on each link (i,j):

$$\underline{\nu}(i,j) = (\nu_0(i,j), \nu_1(i,j), \ldots, \nu_{K-2}(i,j)),$$

where

$$\nu_I(i,j) \triangleq \text{rate at which link } (i,j) \text{, when in aggregate state } I, \text{ receives overflow calls from streams incident to } (i,j).$$

By incident to, we mean having a node in common; that is, (i,k) or (k,j), $k \neq i, j$, are incident to (i,j). Note $\nu_I(i,j) = 0$ if $I \geq K-1$.

In Sections 3.1 and 3.2, we relate these overflow rates to the equilibrium distributions describing the aggregates states of the individual links (i,j):

$$\underline{P}(i,j) = (P_0(i,j), P_1(i,j), \ldots, P_K(i,j))$$

where

$$P_I(i,j) \triangleq \Pr\{\text{link } (i,j) \text{ has aggregate state } I\}.$$

The equations presented in Sections 3.1 and 3.2 define a mapping from overflow rates to overflow rates. Taking the fixed point of this mapping as an estimate of the real overflow rates in the network, we can then easily compute the key measures of steady state performance, in particular, the blocking probability for each stream.

3.1. Overflow Rates

To simplify notation, let us focus on a single node-pair (i,j) and suppress the dependence on (i,j) in the notation whenever possible. There are $M = N-2$ associated alternate paths $\{(i,v),(v,j)\}$ identified by the intermediate or *via* node v. By the link independence assumption, for any $v \neq i,j$,

$$\begin{aligned}\Phi_{v,I} &\triangleq \Pr\{\text{aggregate state of alternate path } \{(i,v),(v,j)\} \text{ is } I\} \\ &= P_I(i,v)\sum_{J=0}^{I} P_J(v,j) + P_I(v,j)\sum_{J=0}^{I-1} P_J(i,v) \qquad (1)\end{aligned}$$

since the aggregate state of the two-link path is the maximum of the aggregate states of the two links. For $v = i$ or j, define $\Phi_{v,I} = 0$ if $I < K$ and $\Phi_{v,K} = 1$. It is convenient to also define

$$\begin{aligned}\Psi_{v,I} &\triangleq \Pr\{\text{aggregate state of alternate path } \{(i,v),(v,j)\} \text{ is } > I\} \\ &= \Phi_{v,I+1} + \ldots + \Phi_{v,K}. \qquad (2)\end{aligned}$$

Now, fix $v \neq i,j$ and $0 \leq I < K-1$, and suppose that alternate $\{(i,v),(v,j)\}$ is in aggregate state I. Under ALBA, this alternate is selected to carry an overflow call if: (i) each of the other alternates $\{(i,u),(u,j)\}$ $(u \neq v)$ has aggregate state $\geq I$, and (ii) alternate $\{(i,v),(v,j)\}$ wins the uniform random selection among those with aggregate state equal to I. Let g_n denote the probability that among the other alternates, n have aggregate state I, and the rest have aggregate states $> I$. Then the probability of selecting the alternate $\{(i,v),(v,j)\}$ to carry an overflow call is

$$\sum_{n=0}^{M-1} \frac{1}{n+1} g_n. \qquad (3)$$

Let us collect the g_n in the generating function:

$$G(x) = \sum_{n=0}^{M-1} g_n x^n. \qquad (4)$$

By the assumption that the aggregate states of the links are independent, the aggregate states of the alternates $\{(i,u),(u,j)\}$ $(u \neq v)$ are independent. Thus, by multiplying the two-point generating functions $x\Phi_{u,I} + \Psi_{u,I}$ $(u \neq v)$, we obtain as the coefficient of x^n the probability that n alternates have aggregate state I and $M-n-1$ have larger aggregate states:

$$G(x) = \prod_{u \neq v} (x\Phi_{u,I} + \Psi_{u,I}). \qquad (5)$$

Integrating (4) from 0 to 1 produces (3), so

$$\int_0^1 \prod_{u \neq v} (x\Phi_{u,I} + \Psi_{u,I}) dx$$

is the probability of selecting the alternate $\{(i,v),(v,j)\}$ to carry an overflow call, and the rate of overflow to this alternate is

$$Y_{v,I} = \lambda(i,j) P_K(i,j) \int_0^1 \prod_{u \neq v} (x\Phi_{u,I} + \Psi_{u,I}) dx,$$

since $\lambda(i,j)P_K(i,j)$ is the rate at which calls arrive to stream (i,j) that cannot be carried direct. Putting the last equation in more general form, and making explicit the dependence on the link identity, we obtain

$$Y_{v,I}(i,j) = \begin{cases} \lambda(i,j)P_K(i,j)\int_0^1 \prod_{u\neq v}(x\Phi_{u,I}(i,j) + \Psi_{u,I}(i,j))dx & \text{if } v \neq i,j \\ 0 & \text{if } v = i \text{ or } v = j. \end{cases} \tag{6}$$

For given I and (i,j), computing the M quantities $Y_{v,I}(i,j)$, $v \neq i,j$, is one of the more delicate and interesting parts of the computation, and is discussed in Section 4.

We are now in position to derive the rates of overflow $\underline{\nu}$ into the links as functions of their aggregate state. Again, focus on a single node-pair (i,j) and suppose its aggregate state is I. Any given incident link (k,i), $k \neq i,j$, assumes aggregate state J with probability $P_J(k,i)$. If link (k,i) is in aggregate state J then:

- the two-link path $\{(k,i),(i,j)\}$ is in aggregate state $I \vee J \triangleq \max\{I,J\}$, and
- stream (k,j) routes overflow calls to this two-link path at rate $Y_{i,I\vee J}(k,j)$.

Summing the contributions of all streams incident to link (i,j) we obtain

$$\nu_I(i,j) = (\sum_{k=0}^{N-1}\sum_{J=0}^{K-2} Y_{i,I\vee J}(k,j)P_J(k,i)) + (\sum_{k=0}^{N-1}\sum_{J=0}^{K-2} Y_{j,I\vee J}(i,k)P_J(k,j)). \tag{7}$$

Note that all quantities on the right hand side are functions of the given parameters and the unknown distributions $\underline{P}(k,i)$ and $\underline{P}(k,j)$, for $k = 0, ..., N-1$.

3.2. Link Distributions

Consider link (i,j), let n denote the link state (number of calls in progress) and I the aggregate state; $n\epsilon\mathcal{A}_I = \mathcal{A}_I(i,j)$, $0 \leq n \leq C = C(i,j)$, $0 \leq I \leq K$. In this state, direct calls arrive to the link at rate $\lambda = \lambda(i,j)$, overflow calls arrive to the link at rate $\nu_I = \nu_I(i,j)$, and calls in progress depart the link at rate n. Thus, the state of the link is a birth-death process, with transition rates

$$\begin{aligned} q_{n,n+1} &= \lambda + \nu_I \quad ; \quad 0 \leq n < C \quad , \quad n\epsilon\mathcal{A}_I \\ q_{n,n-1} &= n \quad ; \quad 0 < n \leq C. \end{aligned}$$

Solving the partial balance equations

$$\pi_n q_{n,n+1} = \pi_{n+1} q_{n+1,n}; 0 \leq n < C \tag{8}$$

and normalizing provides the equilibrium probability π_n of state n. The probability of aggregate state I is given by

$$P_I = \sum_{n\epsilon\mathcal{A}_I} \pi_n.$$

4. Algorithms

In Section 3.1, we derived the overflow rates $\underline{\nu}$ as functions of the link distributions $\underline{P}$. In Section 3.2, we derived the link distributions $\underline{P}$ from the overflow rates $\underline{\nu}$. Viewing the equations as a mapping from overflow rates to overflow rates, a solution is a fixed point. The basic procedure for finding the solution is iteration. That is, given an old tentative solution $\underline{\nu}$ we compute a new tentative solution $\underline{\nu}'$. If the old and new tentative solutions are very close, we stop, accepting $\underline{\nu}'$. Otherwise, we replace the old tentative solution with the new one, and carry out another iteration. Some form of damping may be used to avoid oscillations.

In Section 4.1 we describe the computational model. In Sections 4.2 and 4.3, we describe the details of the computation that takes the link distributions $\underline{P}$ into the overflow rates $\underline{\nu}$, and the computation taking the $\underline{\nu}$ into the $\underline{P}$.

4.1. Computational Model

In order to make a precise and practically meaningful analysis of the parallel algorithms presented below, we focus attention on a particular parallel architecture: the $N \times N$ toroidally connected mesh of processing elements (PEs). Let $PE_{i,j}$ denote the PE at location (i, j) of the mesh, $0 \leq i, j < N$. PE_{ij} is connected to $PE_{i,j+1}$, $PE_{i,j-1}$, $PE_{i+1,j}$, $PE_{i-1,j}$, with the subscript arithmetic modulo N. To implement the fixed point computation, we must first map the data onto the memories of the individual PEs. We assign the data of node-pair (i, j) to two PEs: $PE_{i,j}$ and $PE_{j,i}$. The two PEs compute identical results; in particular, the quantities $\underline{P}(i, j)$ and $\underline{\nu}(i, j)$. Though duplicating the work in this way may seem wasteful, it pays off in very simple code with very efficient interprocessor communication.

We require two basic operations involving interprocessor communication. Suppose that each PE holds a copy of a variable X, with $PE_{i,j}$'s copy denoted $X_{i,j}$. The two operations we require are:

- [Row or Column Broadcast] A row broadcast, given X and a row index i, copies $X_{i,j}$ to the memory of every $PE_{k,j}$, for every $0 \leq k, j < N$. A column broadcast is similar.

- [Row or Column Sum] A row sum, given X and a row index i, stores $\sum_{k=0}^{N-1} X_{k,j}$ into the memory of $PE_{i,j}$, for every $0 \leq j < N$. A column sum is similar.

Thus, in row sums and row broadcasts, the columns act independently; similarly for column sums and column broadcasts.

In the $N \times N$ mesh, a row sum for given X and i can be carried out in $O(N)$ time as follows. At the first step, $PE_{i,j}$ sends $X_{i,j}$ to $PE_{i+1,j}$. At the second step $PE_{i+1,j}$ sends $X_{i,j} + X_{i+1,j}$ to the next PE: $PE_{i+2,j}$. At the third step, $PE_{i+2,j}$ sends $X_{i,j} + X_{i+1,j} + X_{i+2,j}$ to the $PE_{i+3,j}$, and so forth. At the N^{th} step, $PE_{i,j}$ receives the full sum. It is a simple extension to carry out in the same number of steps a row sum for every row $i = 0$, ..., $N - 1$, by circularly shifting N partial sums (one for each i) through each column. Similarly, in $O(N)$ time we can carry out a row broadcast for every row $i = 0$, ..., $N - 1$, by circularly shifting the data through the columns.

Our algorithms have been implemented on a 16384 processor MasPar MP-1 [4]. In this parallel computer the PEs are organized in an 128×128 mesh. The architecture is Single-Instruction-Multiple-Data (SIMD), meaning at each step each active processor executes the same instruction on its local data. Though the individual PEs are very slow, the aggregate peak floating point rate is about 1 Gigaflop. There are two interconnects: a butterflylike [12] circuit-switched network, and a packet switched two dimensional mesh interconnect. The butterflylike network is best suited for general, random communications. On applications, such as ours, which embed naturally into a mesh, communications on the mesh interconnect are much faster. If the problem size $N \times N$ were to exceed the physical machine size $N' \times N'$ ($N' = 128$ on the MasPar) then the larger mesh can be mapped onto the smaller, slowing down the computation by a factor of N/N'.

4.2. Computing Overflow Rates

At each $PE_{i,j}$ the data needed are the distributions $\underline{P}(i,k)$ and $\underline{P}(k,j)$ for all $k \neq i,j$. This data is held in the PEs along the same row and column of $PE_{i,j}$. To initialize the computation across the machine we need only copy each distribution $\underline{P}(i,j)$ from $PE_{i,j}$ to every other PE in the same row and column. This can be done by N row and N column broadcasts in the $N \times N$ mesh. Taking into account that the size of the data $\underline{P}(i,j)$ is $O(K)$, the total time needed is $O(KN)$, using circular shift as described in Section 4.1.

Focus on a single PE, $PE_{i,j}$, and the computation for node-pair (i,j) and aggregate state I. As before, let us suppress (i,j) in the notation, whenever possible. We proceed in two steps:

1. Compute the probabilities $\Phi_{m,I}$ and $\Psi_{m,I}$, by equations (1) and (2), describing the aggregate states of all two-link paths $\{(i,m),(m,j)\}$, $m = 0, \ldots, N-1$.

2. Compute the rates $Y_{m,I}$, describing the flow of overflow calls from (i,j) to each of these two-link paths.

At this point, no interprocessor communication is needed. The first step is straightforward, and takes $O(KN)$ time for all $m = 0, \ldots, N-1$, and all $I = 0, \ldots, K-2$. We next describe two methods for implementing step 2, each having good numerical stability and low computational cost. The methods are based on the integral representation

$$Y_{m,I} = \lambda P_K \int_0^1 H_m(x)dx \text{ where}$$
$$H_m(x) = H(x)/(x\Phi_{m,I} + \Psi_{m,I}),\ H(x) = \prod_{n=0}^{N-1}(x\Phi_{n,I} + \Psi_{n,I}),$$

and take advantage of the commonalities of the integrands H_m for different m. We also implemented "direct" or combinatorial methods for computing the $Y_{m,I}$, but found those implementations were significantly slower and were not stable numerically.

Figure 1 describes the first method, which is based on the observation that the numerical integration of the $M = N-2$ functions $H_m(x)$ is much cheaper if the functions are evaluated at a common set of points x. The time needed to evaluate all M functions at a common point is of the same order, namely $O(N)$, as that needed to compute one of the functions. The total time needed is $O(TN)$, where T is a tunable parameter defining the partition used in the quadrature.

1. Compute and store $H(x_i)$, for $i = 1, ..., T$, where T is a parameter, and the x_i form a partition of $[0, 1]$; e.g., $x_i = i/T$.
2. Estimate each integral, $\int_0^1 H_m(x)dx$, using the values of the function H_m on this partition. Since $H_m(x)$ may increase very rapidly with x, it might be better to use an exponential rather than a linear function (trapezoidal rule) to interpolate between consecutive points of the partition.

Figure 1: Integration via uniform partitioning of $[0, 1]$.

Figure 2 describes another integration method, which relies on an asymptotic expansion of $H_m(x)$ to bring the cost of its numerical evaluation down to $O(1)$ for any m and x. This opens the way to customizing the integration of each H_m to the subinterval of $[0, 1]$ where the mass of H_m is concentrated. Having done so, we can achieve accuracy comparable to the first method with a much coarser partition in the quadrature.

It makes sense to expand $H(x)$ about $x = 1$ since $H(x)$ is increasing with x, assuming its largest value over $[0, 1]$ at 1. Expanding about $x = 1$, it can be shown that

$$H(x) = H(1) \exp\left(\sum_{i=1}^{\infty} \frac{(-1)^{i+1}(x-1)^i}{i} \sum_{n=0}^{N-1} \left(\frac{\Phi_{n,I}}{\Phi_{n,I} + \Psi_{n,I}}\right)^i\right).$$

Truncating the first sum to S terms, where S is a parameter, we obtain as an approximation for $H(x)$,

$$\tilde{H}(x) = H(1) \exp\left(\sum_{i=1}^{S} \frac{(-1)^{i+1}(x-1)^i}{i} \sum_{n=0}^{N-1} \left(\frac{\Phi_{n,I}}{\Phi_{n,I} + \Psi_{n,I}}\right)^i\right), \tag{9}$$

$$\tilde{H}_m(x) = \tilde{H}(x)/(x\Phi_{n,I} + \Psi_{n,I}). \tag{10}$$

Our experience is that for large networks ($N \geq 100$) taking $S = 2$ yields a very accurate estimate $\tilde{H}_m(x)$ for $H_m(x)$. Equations (9) and (10) have the pleasant property that, following an $O(SN)$ time precomputation, evaluating $H_m(x)$ for any m and any x costs just $O(S)$ time. The total time needed for the integration method of Figure 2 is $O(S(T + N))$, where again T is the parameter defining the partition in the quadrature. This completes the discussion of computing the overflow rates $Y_{m,I}$.

1. Compute and store $H(1)$ and $\sum_{n=0}^{N-1}\left(\frac{\Phi_{n,I}}{\Phi_{n,I}+\Psi_{n,I}}\right)^i$, for $i = 1, ..., S$. Using (10) the cost of evaluating any $\tilde{H}_m(x)$ is now $O(S)$.
2. For each m, search $[0, 1]$ for the smallest point $x = x_0(m)$ where $\tilde{H}_m(x)$ is non-negligible. (Our implementation uses binary search, but an interpolation search that exploits the rapid grown of $\tilde{H}_m(x)$ with x would be better.)
3. For each m, estimate $\int_{x_0(m)}^1 \tilde{H}_m(x)dx$ from values $\tilde{H}(x_i(m))$, $i = 1, ..., T$, where the x_i form a partition of $[x_0(m), 1]$ and T is a parameter.

Figure 2: Integration via asymptotic expansion and non-uniform partitioning.

We next compute, for each via k and each aggregate state I, the rate of overflow from stream (i, j) to each row neighbor (i, k), when that neighbor is in aggregate state I,

$$\text{overflow}_I((i,j) \to (i,k)) \triangleq \sum_{J=0}^{K-2} Y_{k,I\vee J}(i,j)P_J(k,j), \tag{11}$$

and the counterpart for each column neighbor (k, j),

$$\text{overflow}_I((i,j) \to (k,j)) \triangleq \sum_{J=0}^{K-2} Y_{k,I\vee J}(i,j)P_J(i,k). \tag{12}$$

$PE_{i,j}$ can compute both sums, for all k and I, in $O(KN)$ time in local memory.

Finally, we sum overflows across the machine to compute the $\nu_I(i, j)$. Using (11) and (12), we may rewrite equation (7) as

$$\nu_I(i,j) = \sum_{k=0}^{N-1} \text{overflow}_I((k,j) \to (i,j)) + \sum_{k=0}^{N-1} \text{overflow}_I((i,k) \to (i,j)). \tag{13}$$

The data of the first sum is held in the column of PEs, $(PE_{k,j})_{k=0}^{N-1}$, and the data of the second sum in the row $(PE_{i,k})_{k=0}^{N-1}$. Thus, we can accumulate the $\nu_I(i, j)$, for all (i, j) by N row and N column summations of the type described in Section 4.1. In the $N \times N$ mesh, the total time needed is $O(N)$. Adding together the costs of all the computations that go into the computation of $\nu_I(i, j)$, for all (i, j) and all I, the time needed is $O(KN)$ with the constants hidden in the $O-$notation dependent on the integration method.

4.3. Computing Link Distributions

As before, focus on a single node-pair (i, j), and the associated processing element $PE_{i,j}$. In practice, $C = C(i, j)$ can be in the thousands, and numerical problems arise if the partial balance equations are solved naively. A simple robust method is described in Figure 3, which scans the process drift to find a state n^* about which the mass will be concentrated, and then telescopes recurrence (8) in both directions about n^* to complete the computation.

1. When in state n the drift, or expected change in state, is $\lambda \cdot 1\{n < C\} + \nu_I - n$, where $n \epsilon \mathcal{A}_I$. Recall that the decreasing sequence of values $n = C$, $C - r_{K-1}$, ..., $C - r_1$, 0 delimit the boundaries of the aggregates. Take n^* to be the largest of these $K+1$ values n such that the drift is nonnegative.

2. Telescope (8) upwards from n^* to determine unnormalized quantities: π_n for $n = n^*+1, ..., C$, and P_I for each aggregate I whose states are $\geq n^*$. Similarly, telescope downwards to obtain the other P_I. Normalize $\underline{P}$.

Figure 3: Computing the distribution $\underline{P}(i,j)$.

The time needed is $O(C)$. Thus, all $\underline{P}(i,j)$ are computed in $O(C_{max})$ time, where $C_{max} \triangleq \max_{(i,j)} C(i,j)$.

5. Experience

We implemented three codes to solve the fixed point equations on a 16384 processor MasPar MP-1 [4]. The codes differ only in the integration technique used to compute the overflow rates $\underline{Y}$:

- [L] uniform partition, linear interpolation (the method of Figure 1)

- [E] uniform partition, exponential interpolation (another variant of the method of Figure 1) and

- [A] non-uniform partition, asymptotic expansion (the method of Figure 2, with linear interpolation.)

The codes were written in MPL, which is based on C. The codes were extensively tested for speed and accuracy against a serial Monte Carlo simulation, and against published data [14] for symmetric networks.

Let us discuss some results on a realistic example, derived from a fiber cut scenario in the AT&T long-distance network. In this example, the number of nodes $N = 114$, the total number of trunks is about 3/4 million. Link capacities and offered loads vary widely from a few tens to a few thousands. As a result of the fiber cut, several hundred links are overloaded and experience significant blocking. In practice, a large fraction of the simulation studies used to assess network performance and reliability investigate similar focused overloads. It turns out that about 13% of the calls are not routed direct, and about 6% are blocked. Figure 4 uses gray-scale to depict the blocking per stream under ALBA with $K = 2$ aggregates, obtained from simulation. Pixel (i,j)'s gray level shows the probability of blocking a call in stream (i,j), interpolating between white (probability 0) and black (probability 1).

In the fixed point approximations, we initially set the overflow rates $\underline{\nu}$ to 0. At each iteration we produce new estimates for the $\underline{\nu}$. At negligible additional cost, at each iteration we compute the aggregate blocking probability,

$$\sum_{(i,j)} \lambda(i,j)\,\mathrm{Pr}\{\text{blocking for stream } (i,j)\} / \sum_{(i,j)} \lambda(i,j),$$

and stopped the computation when the values of this quantity produced by successive iterates differed by less than 10^{-4}. In codes L and E, we evaluated the integrands H_m at $S = 60$ points evenly dividing $[0,1]$. In code A, we used an $S = 2$ term asymptotic expansion $\tilde{H}_m$, and evaluated the $\tilde{H}_m$ at $T = 15$ points evenly dividing the right subinterval of $[0,1]$ where the mass of $\tilde{H}_m$ is concentrated. We found that the numerical differences between the results of the three codes to be negligible, and that the results to be in very good agreement with the simulation (obtained from a 100 million call run, which should be long enough for adequate graphical comparison with the data obtained from the fixed point approximation). A gray-scale plot of the blocking probabilities obtained from the fixed point approximation is indistinguishable from the plot (Figure 4) obtained from simulation. The scatterplot of Figure 5 gives a closer look at the differences. In this figure, for each stream (i,j) with arrival rate greater than 100 erlangs, we plot a point whose x-coordinate is the stream's blocking probability measured in simulation and whose y-coordinate is the stream's blocking probability measured in code L. There were about 1700 such streams, accounting for about 70% of the total traffic. If the simulation and fixed point approximation results were identical, all points would lie on the diagonal.

The performance of the three codes is summarized in Table 1. Codes L and E took 15 iterations to converge and code A took 13 iterations. To gauge speedup, we also implemented C language serial counterparts of codes L and E, and ran the codes on a high performance serial computer: a Silicon Graphics system using the MIPS RS3000 microprocessor, with 128 million bytes of memory. On this system, the serial counterpart of code L took 2.76 hours (177 times slower than the MasPar) and the serial counterpart of code E took 4.84 hours (221 times slower than the MasPar). Taking into account the (very slow) speed of individual PEs on the MasPar parallel computer, speedups between $100x$ and $200x$ over the serial computer are close to the theoretical peak.

Some discussion of the bottlenecks in the codes is in order. We used a general, built-in distributed summing mechanism on the MasPar to implement the row and column summations. Unfortunately, we paid a significant performance penalty, because this mechanism uses the machine's butterflylike circuit-switched interconnect, not the mesh interconnect. Tests show that a customized row and column sum built on the mesh interconnect would run eight times faster, effectively removing this bottleneck. A simple way to speed the integration up by a factor of two would have been to evenly divide the integration task for stream (i,j) evenly between $PE_{i,j}$ and $PE_{j,i}$ instead of having the two PE's each do the complete job. For example, in the uniform partition method (1) one PE could integrate over $[0,1/2)$ and the other over $[1/2,1]$. The overhead needed to communicate, tally, and store the sums at both PEs would be small. Codes E and A owe their relative slowness to the fact that within the integration each evaluation of the integrand involves $\log(\cdot)$ or $\exp(\cdot)$, and we used the system library implementations of these functions. In our application, we do not need the great numerical precision that the library implementations provide. We could speed the integration significantly by using simple, customized versions of $\log(\cdot)$ and $\exp(\cdot)$.

Table 1
Performance of the MasPar codes on the fiber cut example. The percentages are of the total time to convergence.

integration method	# iterations	run time (secs.)	bottlenecks
uniform partition linear interpolation	15	66	row and col. sums (46%) integration (32%)
nonuniform partition asymptotic expansion	13	80	integration (51%) row and col. sums (33%)
uniform partition exponential interpolation	15	143	integration (69%) row and col. sums (22%)

6. Final Remarks

We have have proposed, implemented, and tested efficient massively parallel algorithms for solving the fixed point equations modeling the Aggregated Least Busy Alternative routing policy, for large, asymmetric circuit-switched networks. Very high performance codes were implemented on on a 16384 processor MasPar computer, reducing the time needed to solve a realistic network of over 100 nodes to about a minute, as opposed to a few hours on a high-performance workstation. If more processors are available then greater speedups are possible. In particular, using N^3KC_{max} PEs, the inherent parallelism in the calculations can be exploited to bring the time per iteration down from $O(KN + C_{max})$ to $O(\log N + \log K + \log C_{max})$.

In this paper, we treated the ALBA routing policy, but the same approach applies to any state-dependent routing policy where (i) each link is assigned a *state* as an arbitrary function of the number of calls in progress on the link, (ii) the *state* of an alternate path is defined as a arbitrary function of the states of its links, and (iii) the choice of an alternate path is an arbitrary (random) function of the states of the candidate paths. As long as alternate paths are restricted to two links, the associated system of equations maps naturally onto the mesh, with simple data flows across rows and columns. The computational complexity depends on the defining functions.

What are the implications for the fixed point approach to the approximate analysis of other stochastic models? To date, fixed point approximations have been applied mostly to symmetric models because without symmetry the computational cost of solving the equations can be formidable. On the other hand, as we have demonstrated, the independence assumptions underlying the approximations can lead to tremendous parallelism in the equations, which can be exploited to dramatically reduce the computational burden.

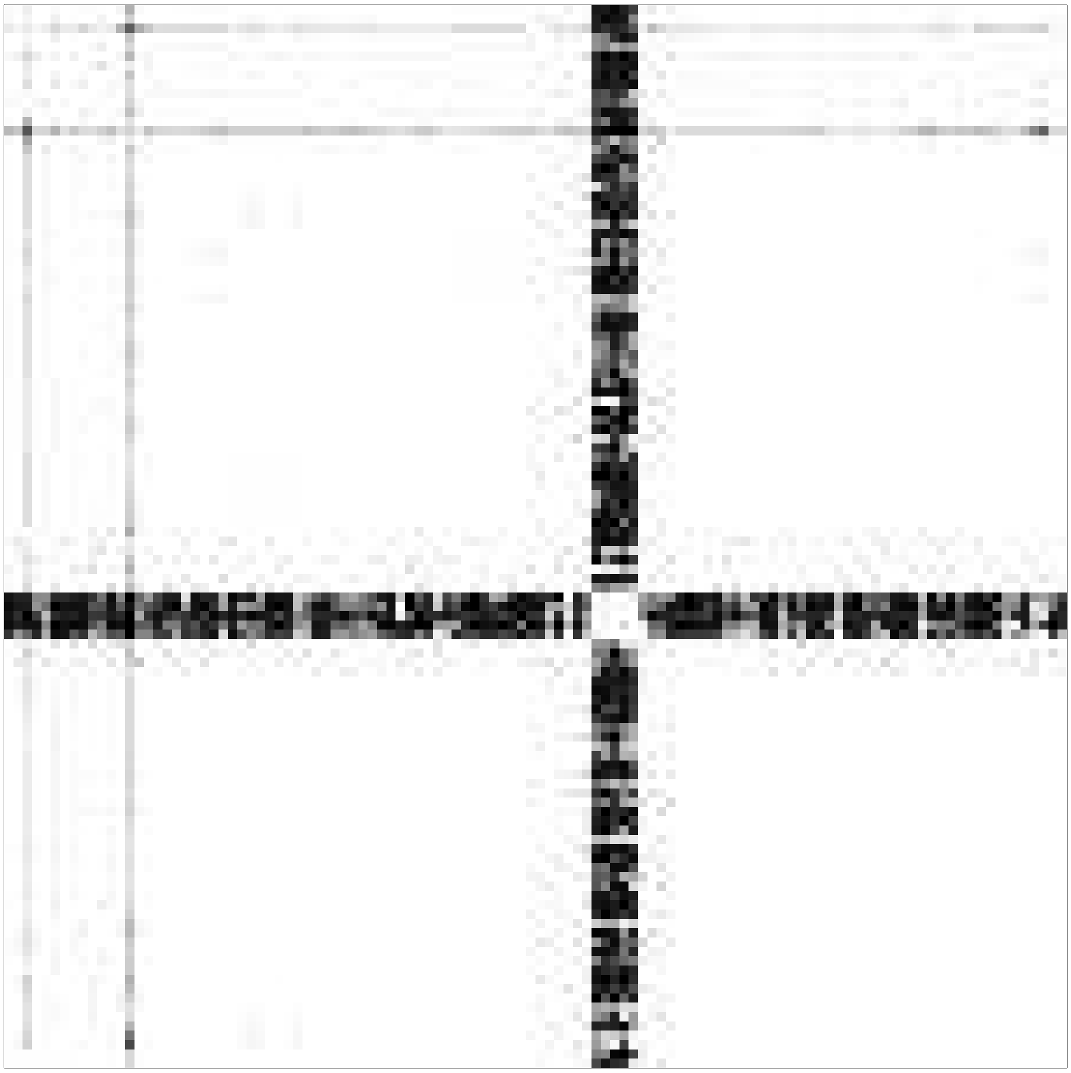

Figure 4: Simulation Results. The outer boarder encloses a symmetric matrix of 114×114 pixels, with pixel (i, j) showing the blocking probability of calls belonging to stream (i, j), for a realistic 114 node network under the ALBA routing policy, with $K = 2$ aggregates. The data were obtained by a 100 million call simulation, in which statistics were gathered over the last 99 million calls. A white pixel corresponds to a link with probability 0 of blocking, and a black pixel to a link with probability 1 of blocking; the gray-level interpolates between these two extremes.

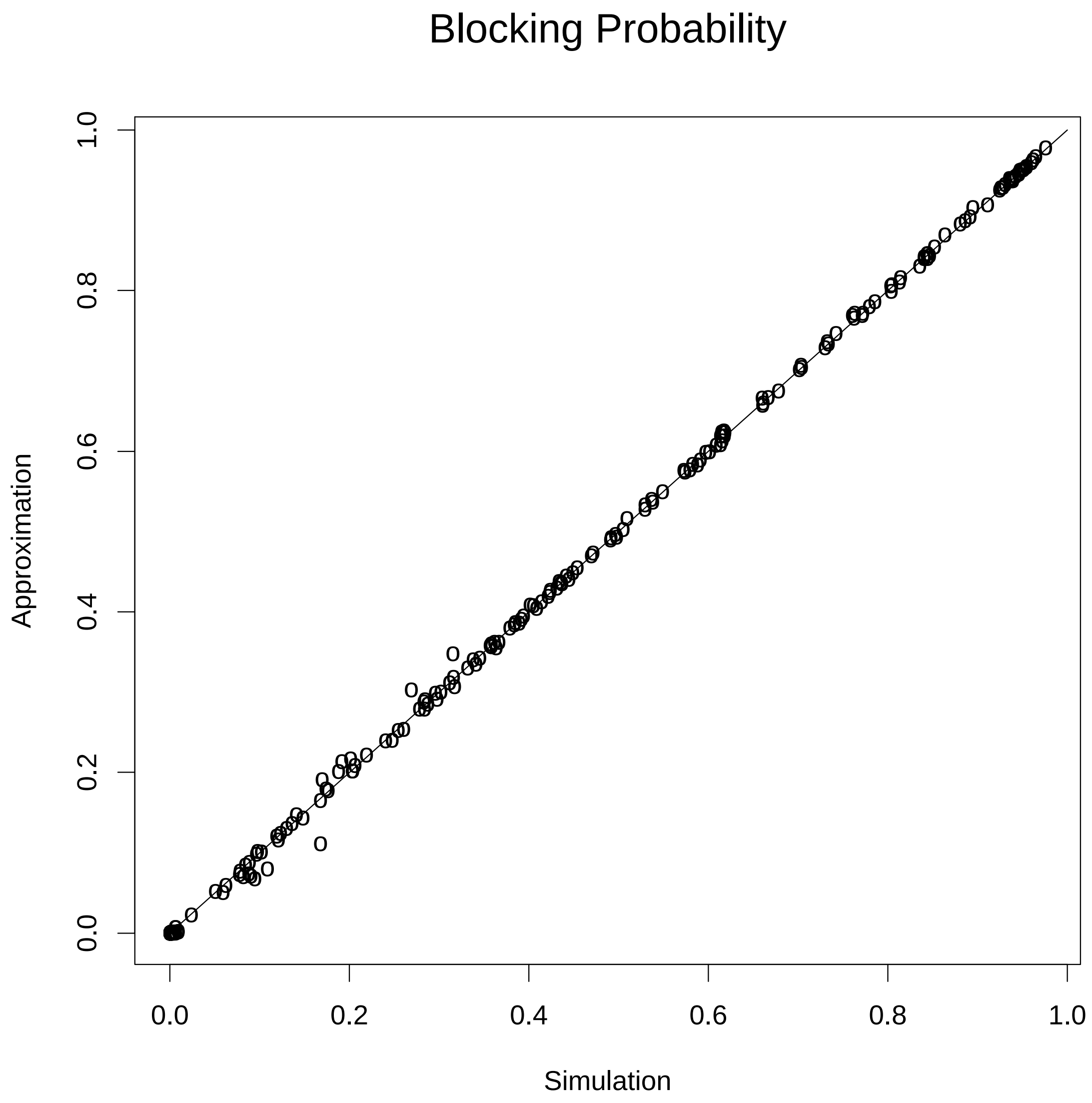

Figure 5: Scatterplot comparing results of the simulation and the fixed point approximation (code L) for each of the roughly 1700 streams with arrival rate exceeding 100 erlangs. For each such stream, we plot a point whose x-coordinate is the blocking probability measured in simulation, and whose y-coordinate is the blocking probability obtained by solving the fixed point equations.

REFERENCES

1. J. M. Akinpelu. The overload performance of engineered networks with nonhierarchical and hierarchical routing. *AT&T Bell Laboratories Technical Journal*, 63:1261–1281, September 1984.
2. G.R. Ash, J.-S. Chen, A.E. Frey, and B.D. Huang. Real-time network routing in a dynamic class-of-service network. In *Thirteenth International Teletraffic Congress (ITC-13)*, Copenhagen, June 1991.
3. G.R. Ash and B.D. Huang. An analytic model for adaptive routing networks. *IEEE Transactions on Communications*, to appear.
4. T. Blank. The MasPar MP-1 architecture. In *Compcon Spring 1990*, San Francisco, CA, February 1990. IEEE Computer Society Press.
5. S.-P. Chung, A. Kashper, and K.W. Ross. Computing approximate blocking probabilities for large loss networks with state-dependent routing. preprint (revised September, 1992), July 1991.
6. A. Girard. Blocking probability of noninteger groups with trunk reservation. *IEEE Transactions on Communications*, COM-33:113–120, February 1985.
7. A. Girard and M.-A. Bell. Blocking evaluation for networks with residual capacity adaptive routing. *IEEE Transactions on Communications*, COM-37:1372–1380, December 1989.
8. S. Katz. Statistical performance analysis of a switched communication network. In *International Teletraffic Congress ITC-5*, pages 566–575, New York, 1967.
9. F.P. Kelly. Blocking probabilities in large circuit-switched networks. *Advances in Applied Probability*, 18:473–505, 1986.
10. F.P. Kelly. Loss networks. *The Annals of Applied Probability*, 1(3):319–378, August 1991.
11. R.S. Krupp. Stabilization of alternate routing networks. In *Proceedings of the IEEE International Conference on Communications*, Philadelpha, 1982. Paper 31.2.
12. F.T. Leighton. *An Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufman, San Mateo, CA, 1992.
13. P.M. Lin, B.J. Leon, and C.R. Stewart. Analysis of circuit switch networks employing originating office control with spill. *IEEE Transactions on Communications*, COM-26:754–765, 1978.
14. D. Mitra, R.J. Gibbens, and B.D. Huang. Analysis and optimal design of aggregated-least-busy-alternative routing on symmetric loss networks with trunk reservations. In *13th International Teletraffic Congress*, Copenhagen, Denmark, June 1991. North Holland.
15. W. Whitt. Blocking when service is required from several facilities simultaneously. *AT&T Technical Journal*, pages 1807–1856, 1985.
16. E.W.M. Wong and T.-S. Yum. Maximum free circuit routing in circuit-switched networks. In *Proceedings of IEEE INFOCOM '90*, pages 934–937, San Francisco, June 1990. IEEE Computer Society Press.