On Bufferless Routing of Variable Length Messages in Leveled Networks

Sandeep N. Bhatt
Bell Communications Research
Morristown, New Jersey

Gianfranco Bilardi
Università di Padova
Padova, Italy

Geppino Pucci
Università di Padova
Padova, Italy

Abhiram Ranade
University of California
Berkeley, California

Arnold L. Rosenberg
University of Massachusetts
Amherst, Massachusetts

Eric J. Schwabe
Northwestern University
Evanston, Illinois

Abstract

We study the most general communication paradigm on a multiprocessor, wherein each processor has a distinct message (of possibly distinct lengths) for each other processor. We study this paradigm, which we call *chatting*, on multiprocessors that do not allow messages once dispatched ever to be delayed on their routes. By insisting on *oblivious* routes for messages, we convert the communication problem to a pure scheduling problem. We introduce the notion of a *virtual chatting schedule*, and we show how efficient chatting schedules can often be produced from efficient virtual chatting schedules. We present a number of strategies for producing efficient virtual chatting schedules on a variety of network topologies.

Authors' Mailing Addresses:

S. N. Bhatt: Bell Communications Research, 435 South St., Morristown, NJ 07960

G. Bilardi: Dipartimento di Elettronica e Informatica, Università di Padova, 35131 Padova, ITALY

G. Pucci: Dipartimento di Elettronica e Informatica, Università di Padova, 35131 Padova, ITALY

A. Ranade: Computer Science Division, University of California, Berkeley, CA 94720

A. L. Rosenberg: Department of Computer Science, University of Massachusetts, Amherst, MA 01003

E. J. Schwabe: Dept. of Electrical Engineering and Computer Science, Northwestern Univ., Evanston, IL 60201

1 Introduction

Efficient interprocessor communication is recognized as one of the most challenging aspects of parallel computing. In this paper, we study the most general modality of interprocessor communication within a network of processors:[1] *all-to-all personalized communication*, or, for short, *chatting*. This modality is characterized by the network's PEs passing among themselves arbitrary patterns of messages of arbitrary lengths. In particular, each PE is allowed to send possibly distinct messages of possibly distinct lengths to any subset of the other PEs. This modality of communication is computationally more demanding than its more commonly studied relatives, which typically restrict the number of messages a PE can send or receive in a single operation and/or the variability of the contents or lengths of messages.

This paper is devoted to studying how to schedule the transmission of messages in a chatting operation in a way that minimizes the total time for the operation, within the context of the following conventions.

- **Communication in rounds.** Our processor networks operate in alternating phases of local (or, intra-PE) computation and global (or, inter-PE) communication.

This regimen, which is consistent with the philosophy underlying the bulk-synchronous computation model [14], somewhat simplifies the design of programs for a processor network, in a manner that is expounded on at length in that source.

- **Message integrity.** Each message in a chatting operation travels through the network as a contiguous stream of flits.[2]

This convention, which minimizes the amount of addressing information that must travel throughout the network along with the messages, is also a characteristic of wormhole routing [3, 5, 10].

- **Oblivious Routing.** Each message in a chatting operation travels through the network along a fixed path which is determined solely by the source and destination PEs of the message.

[1] A *processor network* (or, *processor array*) is a parallel architecture comprising a set of identical processing elements (PEs) that communicate across an interconnection network.

[2] A *flit* is the largest unit of information that can be transmitted along a PE-to-PE link in one communication step.

This convention, which contrasts with typical studies of packet routing (see, e.g., [13]) and wormhole routing, reduces our message routing problems to pure scheduling problems. The many respects in which our computing model differs from that of [2] renders the conclusions in that paper about the inevitable inefficiency of oblivious routing irrelevant for our study.

- **Bufferless communication.** The PEs of our networks have neither buffers nor queues to store messages being relayed through them; hence messages, once dispatched, are never delayed en route.

Our demand for unbuffered communication converts the chatting problem to a pure scheduling problem. This demand is shared with the regimen of hot potato (or, deflection) routing [4, 9]; it appears also in the study of scattering and gathering general messages in general networks, in [1]. One might be able to rationalize this demand in terms of resource conservation: buffering requires both additional memory (each PE must be prepared to store the longest message in the system) and time (e.g., for the processing of addresses). However, our overriding motivation in this study was to understand communication in networks better, by determining the cost of this strict assumption in terms of the complexity of the problem of chatting.

The confluence of all of these conventions — in particular, our focus on communication in rounds, on oblivious routing of messages, and on general patterns of passing arbitrarily many messages of arbitrary lengths — makes the object of our study rather different from other studies of communication in processor networks. One major difference should be stressed here. We assume that the one (generic) communication phase we concentrate on is characterized by a set of messages that the PEs want to exchange with one another; new messages are not introduced into the system during the phase. Our goal is to devise (scheduling) algorithms that will route the messages efficiently to their designated destinations without any buffering.

A Roadmap. The remainder of the paper is organized as follows.

Section 2 introduces the formal setting for our study. We develop there the notions of a chatting schedule for a set of messages in a network of processors and of the duration of a chatting schedule — the quantity we strive to minimize.

In Section 3, we introduce the notion of a virtual chatting schedule for a set of messages in a network of processors and of the duration of a virtual chatting schedule. We show that, for the class of "forward-leveled" networks, one can convert an efficient virtual chatting schedule (which are often easier to develop than are efficient chatting schedules) into an almost equally efficient chatting schedule.

The remaining sections build on the transformation technique of Section 3. They focus on designing efficient virtual chatting schedules which will later be converted to efficient "ordinary" chatting schedules.

Section 4 is devoted to showing how to use known solutions to the so-called rectangle compaction problem to create virtual chatting schedules for linear array networks, that are within a small constant factor of optimal. The techniques of Section 3 can then be used to convert these efficient virtual chatting schedules into almost equally efficient chatting schedules.

In Section 5, we study how to devise efficient virtual chatting schedules that are based on the set of lengths of the messages to be transmitted. We present a scheduling strategy that reduces the problem of routing messages of arbitrary length to the problem of routing unit-length messages. We apply this strategy in Section 5.2 to chatting problems on two-dimensional mesh networks.

Section 6 explores an approach to message scheduling that is based on decomposing the host network by suitable cuts. This approach is then shown to yield efficient chatting schedules for tree networks.

2 The Problem Formalized

2.1 Processor Networks

As is customary, we identify a processor network with a directed graph $G = (V, E)$ whose nodes V represent the network's PEs and whose arcs E represent its communication links. The source of an arc is an *output port* of the incident PE; the destination of an arc is an *input port* of the incident PE. In a less customary way, we associate with each processor network G an *atlas* that designates a path $\rho(u, v)$ from every node $u \in V$ to every node $v \in V$ that is accessible from u; any message from node u to node v is routed along path $\rho(u, v)$. (This means that all message passing is *oblivious* in the sense that the route of a message is determined entirely by the message's source and destination nodes.)

Our processor networks operate in a *pulsed* fashion: a network alternates *computation phases*, in which (in parallel) each PE performs some computation, and *communication phases*, in which messages flow among the PEs. The networks operate synchronously, at least during the communication phases. More specifically, during each step of a communication phase, all PEs (in parallel) perform the following actions:

- send at most one flit along each outgoing arc,

- receive at most one flit along each incoming arc,
- compute whether and where to send flits during the next step.

The PEs in a processor network $G = (V, E)$ *do not have message buffers.* In particular, this means that if, at time t, a PE $u \in V$ receives a flit that is not destined for it, then u sends that flit out toward its destination at time $t + 1$.

2.2 The Chatting Problem

A *message* is a sequence $M = \langle m_0, m_1, \ldots, m_{\ell-1} \rangle$ of flits; we call ℓ the *length* of message M. An *addressed message in processor network* $G = (V, E)$ is a message M, together with a designated source PE $u \in V$ and a designated destination PE $v \in V$ (hence with a designated path $\rho(u, v)$ in G).

A *chatting problem for processor network* $G = (V, E)$ is a set of addressed messages in G.[3] This paper is devoted to studying algorithms that "solve" a chatting problem by scheduling the transmission of all messages so that they get from their designated sources to their designated destinations without ever encountering contention for either a PE or a communication link, i.e, so that they honor the oblivious, bufferless communication regimen we are studying. In addition to these restrictions on our chatting algorithms, we insist that messages travel as *indivisible units*, i.e., that at any moment, all flits of an addressed message that do not currently reside in either the message's source or destination node occupy a contiguous path of arcs in G, with one message-flit per path-arc.

2.3 Chatting Schedules

A *chatting schedule* for a chatting problem $\mathcal{M}$ on a processor network G is a function τ that associates an integer (time) with each flit m of an addressed message $M \in \mathcal{M}$ and each arc e on the designated path from the source to the destination of M. The interpretation is that flit m traverses arc e at step $\tau(m, e)$ of the communication phase.

A. Admissible Chatting Schedules

In order to comply with all of our assumptions about bufferless transmission of indivisible messages, a chatting schedule τ for a chatting problem $\mathcal{M}$ must satisfy the

[3] A purist might argue that a set of addressed messages in G is really an *instance* of the chatting problem. We abuse terminology to simplify exposition.

following constraints:

- *Conflict-free Arcs.* At most one flit traverses any given arc at any given time:

$$[\tau(m,e) = \tau(m',e)] \Rightarrow [m = m']. \tag{2.1}$$

- *Bufferless nodes.* Once transmitted from its source PE, a flit m moves "forward" one arc at every step: if $\langle e_0, e_1, \ldots, e_{d-1}\rangle$ is the designated path for the message that m belongs to, then, for $k = 0, 1, \ldots, d-1$,

$$\tau(m, e_k) = \tau(m, e_0) + k. \tag{2.2}$$

- *Message integrity.* Flits of the same message $\langle m_0, m_1, \ldots, m_{\ell-1}\rangle$ traverse a given arc e at consecutive times: for $h = 0, 1, \ldots, \ell - 1$,

$$\tau(m_h, e) = \tau(m_0, e) + h. \tag{2.3}$$

We call a chatting schedule that honors these restrictions *admissible*. Henceforth, we consider only admissible chatting schedules.

Remark. Let τ be a chatting schedule for the chatting problem $\mathcal{M}$. The schedule that τ assigns to the addressed message $M = \langle m_0, m_1, \ldots, m_{\ell-1}\rangle \in \mathcal{M}$, along its designated path $\rho = \langle e_0, e_1, \ldots, e_{d-1}\rangle$ is determined uniquely by the value $\tau(m_0, e_0)$ and the quantities d and ℓ; to wit, for all $0 \leq h < \ell$ and all $0 \leq k < d$,

$$\tau(m_h, e_k) = \tau(m_0, e_0) + h + k. \tag{2.4}$$

B. Efficient Chatting Schedules

The *duration* $T(\tau)$ of a chatting schedule τ for a chatting problem $\mathcal{M}$ is the amount of time it takes to deliver all messages in $\mathcal{M}$; formally:

$$T(\tau) = \max_{m,e}\{\tau(m,e)\} - \min_{m,e}\{\tau(m,e)\} + 1, \tag{2.5}$$

where the minimization and maximization are over all flits m in the chatting problem $\mathcal{M}$ and all arcs e that occur in the designated paths for the addressed messages in $\mathcal{M}$.

The objective of our study is to devise techniques for constructing admissible chatting schedules with (close to) minimum duration.

2.4 Two Significant Observations

A. Centralized vs. Distributed Schedules

One's intuition might suggest that the only efficient chatting schedules are those that are totally *distributed*, in the sense of not requiring the intervention of a central master PE. Yet, if one is willing to suffer a modest amount of time for the scheduling — equal, say, to a small multiple of the number of PEs in the underlying network G — then one can design efficient "semi-distributed" efficient chatting schedules as follows.

1. Using standard techniques in the field of parallel algorithmics, one has the PEs in the network elect a leader, call it P_0.

2. Using the efficient (distributed) gathering schedules in [1], the PEs in G transmit to PE P_0 a "summary" of chatting problem $\mathcal{M}$. The summary represents each addressed message $M \in \mathcal{M}$ by the triple $\langle$source of M, destination of M, length of $M\rangle$. Note that the summary comprises a fixed number of flits, irrespective of the length of M.

3. PE P_0 uses the summary to compute, off-line, an efficient chatting schedule for $\mathcal{M}$.

4. Using the optimally efficient scattering schedules in [1], PE P_0 scatters to each of the other PEs a message transmission schedule. The amount of message traffic in this scattering operation is minimized if the chatting schedule has each PE dispatch all of its messages in a gap-free stream.

Although some of our algorithms for producing chatting schedules operate in a distributed manner, the preceding algorithmic strategy suggests that even centralized algorithms are of interest, given the algorithms for optimal scattering and efficient distributed gathering in [1].[4]

B. Two Simple Lower Bounds

Certain simple parameters of a chatting problem $\mathcal{M}$ yield straightforward, but useful, lower bounds on the duration $T(\tau)$ of any chatting schedule τ for $\mathcal{M}$.

Denote by $\ell(M)$ the length (number of flits) of addressed message $M \in \mathcal{M}$; denote by $d(M)$ the distance (number of arcs) traversed by M along its designated path; denote by

[4]The strategy in this subsection must be evaluated in the light of the restriction in [1] to processor networks that observe the *single-port communication regimen*, wherein, in a single step, each PE can send at most one flit along at most one outgoing arc and receive at most one flit along at most one incoming arc.

$C(e)$ the number of message-flits from $\mathcal{M}$ whose designated path contains arc e. ($C(e)$ is called the *arc congestion* of the chatting problem.) Define:

$L(\mathcal{M}) = \max\{\ell(M) : M \in \mathcal{M}\}$	maximum message length in $\mathcal{M}$
$D(\mathcal{M}) = \max\{d(M) : M \in \mathcal{M}\}$	maximum routing-path length for any $M \in \mathcal{M}$
$Q(\mathcal{M}) = \max\{\ell(M) + d(M) - 1 : M \in \mathcal{M}\}$	longest transit time for any $M \in \mathcal{M}$
$C(\mathcal{M}) = \max\{C(e) : e \in E\}$	maximum arc congestion for problem $\mathcal{M}$

We obtain the following lower bounds.

Proposition 2.1 *For any chatting schedule τ for a chatting problem $\mathcal{M}$,*

$$T(\tau) \geq \max(C(\mathcal{M}), Q(\mathcal{M})).$$

Proof: Observing that transmitting a message M from its source to its destination takes $\ell(M) + d(M) - 1$ steps, we have $T(\tau) \geq Q(\mathcal{M})$. Observing that an admissible schedule must keep arcs conflict free (condition (2.1)), we have $T(\tau) \geq C(\mathcal{M})$. □

2.5 Unidirectional Linear Arrays: A Running Example

To illustrate the various notions we have introduced and shall be studying, it is helpful to refer repeatedly to a simple specific network. We introduce for this purpose the *unidirectional linear array* (ULA), whose node set is $V = \{0, 1, \ldots, N-1\}$ and whose arc set is $E = \{(0,1), (1,2), \ldots, (N-2, N-1)\}$. The ULA is a convenient platform for gaining intuition about our message-scheduling problem, both because of its inherent simple structure and because every chatting schedule τ for a ULA admits the following convenient geometric representation, that originates in [1].

We consider the positive quadrant of the integer plane as the processor-time plane for a ULA: abscissa x corresponds to the ULA-arc $\eta_x \stackrel{\text{def}}{=} (x, x+1)$, while ordinate t corresponds to discrete time-step t. Given a chatting schedule τ for a chatting problem $\mathcal{M}$, we say that integer point (x,t) in the plane is *marked* if, for some flit m within $\mathcal{M}$, $\tau(m, \eta_x) = t$; point (x,t) is *unmarked* otherwise. Because of Conditions (2.2) and (2.3) on τ, the marked region corresponding to any single message $M \in \mathcal{M}$ is a *parallelogram* with two sides parallel to the vertical (time) axis and the other two parallel to the main bisector (see Figure 1). The parallelogram mirrors in a natural way the transition of the successive flits of message M along the PEs of the ULA. If we now consider the set of parallelograms that describe the transit of all of the messages in $\mathcal{M}$, condition (2.1) guarantees — in fact, is equivalent to — the condition that parallelograms corresponding to distinct messages do not overlap.

This geometric interpretation of message transmission in a ULA reduces the problem of finding a minimum-duration chatting schedule for a chatting problem on a ULA to the following geometric problem.

Parallelogram Compaction.
Given: A set of parallelograms in the integer plane (with sides as described above), each having a fixed projection on the horizontal (ULA) axis but being free to slide up or down in the vertical (time) direction,
Find: A pairwise nonoverlapping placement of the parallelograms that minimizes the difference between the maximum and minimum ordinates of any of their points.

We shall return to this example repeatedly.

3 Virtual Chatting Schedules

In this section, we introduce *virtual* chatting schedules, and we study their relation with "ordinary" chatting schedules. We show, by example, that devising an efficient virtual chatting schedule for a problem is sometimes easier than devising an efficient chatting schedule. Moreover, we specify a class of networks for which one can convert an efficient virtual chatting schedule into an efficient chatting schedule.

3.1 The Notion of Virtual Schedule

A *virtual chatting schedule* for a chatting problem $\mathcal{M}$ on a processor network G is a function σ that associates an integer (time) with each flit m of an addressed message $M \in \mathcal{M}$ and each arc e on the designated path from the source to the destination of M. In analogy with our concentration on admissible chatting schedules, we insist that every virtual chatting schedule σ satisfy certain constraints; in contrast to the notion of admissibility, these constraints do not necessarily correspond to either architectural or algorithmic features.

- *Arc conflict-free.* At most one flit traverses a given arc at any given time:
$$[\sigma(m, e) = \sigma(m', e)] \Rightarrow [m = m']. \tag{3.1}$$
- *Message "transmission" and integrity.* Every flit of a message $M = \langle m_0, m_1, \ldots, m_{\ell-1} \rangle$ "traverses" all the arcs in M's designated path $\langle e_0, e_1, \ldots, e_{d-1} \rangle$ at the same time. Formally, for all $h = 1, 2, \ldots, \ell - 1$ and all $k = 1, 2, \ldots, d - 1$,
$$\sigma(m_h, e_k) = \sigma(m_0, e_0) + h. \tag{3.2}$$

When we discuss virtual chatting schedules, it is often convenient to subdivide the message "transmission" and integrity constraint (3.2) into its two logical constituents:

- *Instantaneous transmission*

$$\sigma(m, e_k) = \sigma(m, e_0). \tag{3.3}$$

- *Virtual message integrity.*

$$\sigma(m_h, e) = \sigma(m_0, e) + h. \tag{3.4}$$

The "physical" interpretation of a virtual chatting schedule is less obvious than is that of an "ordinary" chatting schedule. The easiest interpretation views a virtual chatting schedule as a mechanism that reserves paths in G for contiguous blocks of time. Specifically, a virtual schedule σ reserves the entire designated path for each message $M \in \mathcal{M}$ for the entire block of time when message M is in transit from its source node to its destination node. This interpretation is consistent with constraint (3.2).

Remark. Let σ be a virtual chatting schedule for the chatting problem $\mathcal{M}$. By dint of constraint (3.2), the (virtual) schedule that σ assigns to the addressed message $M = \langle m_0, m_1, \ldots, m_{\ell-1} \rangle \in \mathcal{M}$, along its designated path $\rho = \langle e_0, e_1, \ldots, e_{d-1} \rangle$ is determined uniquely by the value $\sigma(m_0, e_0)$ and the quantities d and ℓ.

3.2 The Duration of a Virtual Chatting Schedule

In analogy with the duration of a chatting schedule, one can define the duration of a virtual chatting schedule.

The *duration* $S(\sigma)$ of a virtual chatting schedule σ is the quantity

$$S(\sigma) = \max_{m,e}\{\sigma(m, e)\} - \min_{m,e}\{\sigma(m, e)\} + 1, \tag{3.5}$$

where the minimization and maximization are over all flits m in the chatting problem $\mathcal{M}$ and all arcs e that occur in the designated paths for the addressed messages in $\mathcal{M}$.

Also in analogy with chatting schedules, one can derive simple lower bounds on the duration of virtual chatting schedule of the sort provided by Proposition 2.1.

Proposition 3.1 *For any virtual chatting schedule σ for a chatting problem $\mathcal{M}$,*

$$S(\sigma) \geq \max(C(\mathcal{M}), L(\mathcal{M})).$$

Proof: Constraint (3.2) mandates that $S(\sigma) \geq L(\mathcal{M})$. The requirement that arcs be conflict free (Constraint (3.1)) forces the inequality $S(\sigma) \geq C(\mathcal{M})$. □

3.3 Unidirectional Linear Arrays: A Running Example

A virtual chatting schedule for a ULA admits a geometric representation similar to the one developed for a chatting schedule in Section 2.5. In the present, virtual scenario, the marked region associated with a given message is an *isothetic rectangle*, i.e., a rectangle with sides parallel to the coordinate axes (see Figure 2).

In analogy with the geometric reduction noted in Section 2.5, the geometric representation of virtual chatting schedules reduces the problem of finding a virtual chatting schedule of minimum duration to the following purely geometric problem.

Rectangle Compaction.
Given: A set of isothetic rectangles in the integer plane, each having a fixed projection on the horizontal (ULA) axis but being free to slide up or down in the vertical (time) direction,
Find: A pairwise nonoverlapping placement of the rectangles that minimizes the difference between the maximum and minimum ordinates of any of their points.

3.4 Comparing Ordinary and Virtual Chatting Schedules

For the class of *forward-leveled networks*, there is an intimate relationship between ordinary and virtual chatting schedules.

A *forward-leveled network* (*FL-network*, for short) is one whose underlying directed graph $G = (V, E)$ is *forward-leveled*, in the following sense. The node-set V of G admits a partition $V = V_0 + V_1 + \cdots + V_H$ in such a way that, for every arc $e \in E$, there is an i such that $e \in V_i \times V_{i+1}$; in this case, we say that level$(e) = i$. In the partition of V, we call block V_i the ith *level* of G. Note that in an FL-network, for each chatting path $\langle e_0, e_1, \ldots, e_{d-1} \rangle$, level$(e_k)$ = level$(e_{k-1}) + 1$, for $k = 1, 2, \ldots, d-1$.

While FL-networks form a narrow class of networks, the results that we obtain for this class can be applied to broader classes of networks in a variety of ways:

- One can emulate arbitrary directed acyclic networks using FL-networks.
- One can decompose (a spanning subnetwork of) one's general network into FL-networks.

An instance of the second transformation would be the decomposition of a *bidirectional* linear array (in which every pair of adjacent nodes, say i and $i+1$ are connected by two opposing arcs, $(i, i+1)$ and $(i+1, i)$) into two ULAs. Such indirect use of a restricted

class of networks to implement communication primitives in broader classes can be found also in [1], where one implements general scattering and gathering operations in spanning trees, in [8], where one implements broadcasting operations in spanning trees, and in [11], where one distributes synchronization tokens in specially chosen spanning subgraphs.

We now expose some relations between ordinary and virtual chatting schedules on FL-networks. Each relation builds on a method of transforming either a virtual chatting schedule into an ordinary one, or vice versa; each carries with it a performance guarantee on the derived schedule.

A. A Local Symmetric Transformation

We begin with a transformation that converts either of a virtual or ordinary chatting schedule into the other. The duration of the derived schedule is bounded in a simple way in terms of the duration of the original schedule. The transformation employs no global information about the original schedule.

The LS Transformation.

Let $\mathcal{M}$ be a chatting problem on the FL-network G. Consider functions τ and σ, each of which associates an integer with each flit m of an addressed message $M \in \mathcal{M}$ and each arc e on the designated path from the source to the destination of M. Call any such function a *pseudo-schedule* for $\mathcal{M}$. If τ and σ are related by the equations

$$\tau(m, e) = \sigma(m, e) + \text{level}(e) \tag{3.6}$$

for all flits m and arcs e, then we say that they are *LS-related.*

Theorem 3.1 *Let $\mathcal{M}$ be a chatting problem on the $(H+1)$-level FL-network G; let τ and σ be pseudo-schedules for $\mathcal{M}$ that are LS-related.*
(a) *The pseudo-schedule τ is a chatting schedule for $\mathcal{M}$ if and only if the pseudo-schedule σ is a virtual chatting schedule for $\mathcal{M}$.*
(b) *If τ and σ are, respectively, a chatting schedule and a virtual chatting schedule for $\mathcal{M}$, then*

$$|T(\tau) - S(\sigma)| \leq H - 1. \tag{3.7}$$

Proof: **(a)** The functional relation (3.6) between τ and σ implies that, for all arcs e and all flits m and m', $\tau(m, e) = \tau(m', e)$ if and only if $\sigma(m, e) = \sigma(m', e)$. Therefore, τ avoids conflicts on arcs (constraint (2.1)) if, and only if, σ does (constraint (3.1)).

The bufferless-node constraint (2.2) on τ combines with the functional relation (3.6) to yield

$$\sigma(m, e_k) + \text{level}(e_k) = \sigma(m, e_0) + \text{level}(e_0) + k.$$

Since level(e_k) = level(e_0)+k, this equation implies that σ has instantaneous transmission (constraint 3.3). By the same reasoning, if σ satisfies constraint (3.3), then relation (3.6) implies that τ satisfies the bufferless-node constraint (2.2).

The message-integrity constraint (2.3) on τ combines with the functional relation (3.6) to verify that σ has virtual message integrity (constraint 3.4). This constraint combines with σ's instantaneous transmission (property (3.3)) to verify that σ enjoys property (3.2). By the same reasoning, if σ enjoys property (3.2), then relation (3.6) implies that τ satisfies the message-integrity constraint (2.3).

In summation, we have shown that τ satisfies the constraints of a chatting schedule if, and only if, σ satisfies the constraints of a virtual chatting schedule.

Figure 3 illustrates relation (3.6) between τ and σ on a ULA when both are schedules. This should lend some intuition for part (b) of the proof.

(b) Say that $\tau(m_1, e_1) = \max_{m,e}\{\tau(m, e)\}$ and that $\tau(m_2, e_2) = \min_{m,e}\{\tau(m, e)\}$, where the maximization and minimization are as in definition (2.5). Then, starting with that definition,

$$\begin{aligned} T(\tau) &= \tau(m_1, e_1) - \tau(m_2, e_2) + 1 \\ &= \sigma(m_1, e_1) + \text{level}(e_1) - \sigma(m_2, e_2) - \text{level}(e_2) + 1 \\ &\leq |\sigma(m_1, e_1) - \sigma(m_2, e_2)| + |\text{level}(e_1) - \text{level}(e_2)| + 1 \\ &\leq S(\sigma) + H - 1. \end{aligned}$$

By a symmetric argument, one shows that $S(\sigma) \leq T(\tau) + H - 1$. □

Inequality (3.7) can be viewed as a performance guarantee on the chatting schedule τ_σ that is obtained from the virtual chatting schedule σ via relation (3.6). In certain cases, say when $S(\sigma)$ is considerably smaller than H, the bound on $T(\tau_\sigma)$ inferred from inequality (3.7) may be rather loose, even if σ is an optimal virtual chatting schedule. In the next subsection, we present transformations between ordinary and virtual chatting schedules that are guaranteed to preserve quality better than the LS transformation (3.6) does.

B. Two Global Transformation

In this subsection, we present two transformations, one that produces an ordinary chatting schedule from a given virtual one, the other that performs the converse transformation. Both of these transformations have better performance guarantees than that given by inequality (3.7). As the subsection title suggests, the improved guarantees are possible because the transformations utilize some global information about the original schedule in defining the derived one; in both cases, the global information resides in the

duration of the original schedule. For both transformations, we first verify that they do indeed produce the desired type of schedule, with the advertised performance guarantee. We then illustrate the transformation and the guarantee for the case of a ULA, where the graphical representation affords a better intuitive grasp of the ideas.

The VO Transformation. Our first transformation produces (ordinary) chatting schedules from virtual chatting schedules. The derived chatting schedules are "slower" than their originating virtual chatting schedules by no more than the longest transit time of any message in the chatting problem.

Say that we are given a virtual chatting schedule σ, of duration $S(\sigma)$, for the chatting problem $\mathcal{M}$ on the FL-network G. Letting $\lambda(e) = \text{level}(e) - (\text{level}(e) \bmod S(\sigma))$ for each arc e of G, define the *VO-derived pseudo-schedule* τ_σ as follows.[5] In order to evaluate $\tau_\sigma(m,e)$ for a given flit m belonging to some message $M \in \mathcal{M}$ and a given arc e of G, one looks at the first flit, call it m_0, of message M and at the first arc, call it e_0, of the designated path of message M. One then assigns

$$\tau_\sigma(m,e) \stackrel{\text{def}}{=} \begin{cases} \sigma(m,e) + \text{level}(e) - \lambda(e_0) & \text{if } \sigma(m_0,e_0) \leq S(\sigma) - (\text{level}(e_0) \bmod S(\sigma)) \\ \sigma(m,e) + \text{level}(e) - \lambda(e_0) - S(\sigma) & \text{otherwise.} \end{cases} \tag{3.8}$$

Theorem 3.2 (VO-derived chatting schedules)
Let σ be a virtual chatting schedule of duration $S(\sigma)$, for the chatting problem $\mathcal{M}$ on the FL-network G; let τ_σ be the pseudo-schedule that is VO-derived from σ. Then τ_σ is a chatting schedule for $\mathcal{M}$ of duration

$$T(\tau_\sigma) < S(\sigma) + Q(\mathcal{M}). \tag{3.9}$$

Proof: Assume, with no loss of generality, that σ satisfies $1 \leq \sigma(m,e) \leq S(\sigma)$ for all flits m belonging to problem $\mathcal{M}$ and all arcs e of G. (This is just a normalization of the schedule.)

We verify first that τ_σ is a chatting schedule. Since σ is a virtual chatting schedule, one can invoke the message transmission and integrity property (3.2) to verify that τ_σ enjoys both bufferless nodes (2.2) and message integrity (2.3). It remains only to show that τ_σ does not allow arc conflicts. To this end, say that there exist flits m and m' and an arc e such that $\tau_\sigma(m,e) = \tau_\sigma(m',e)$. Noting that flits m and m' may come from different addressed messages in $\mathcal{M}$, so that the two occurrences of e may be associated with distinct paths having distinct initial arcs, call them e_0 and e_0', we find directly from

[5] Of course, τ_σ is a chatting schedule, not just a pseudo-schedule; but this must be verified before we can legitimately assert it.

definition (3.8) that

$$(\sigma(m,e) - \sigma(m',e)) + (\lambda(e'_0) - \lambda(e_0)) = \alpha S(\sigma),$$

for some $\alpha \in \{-1, 0, 1\}$. If we take each side of this equation modulo $S(\sigma)$, and we observe that, for any arc e, $\lambda(e) \bmod S(\sigma) = 0$, then we find that

$$\sigma(m,e) \bmod S(\sigma) = \sigma(m',e) \bmod S(\sigma).$$

Given that $1 \leq \sigma(m,e), \sigma(m',e) \leq S(\sigma)$, the latter equality implies that

$$\sigma(m,e) = \sigma(m',e).$$

Since σ is arc-conflict free (being a virtual chatting schedule), we must have $m = m'$. It follows that τ_σ is arc-conflict free, hence, finally, is a chatting schedule.

Next, let us compare the durations $T(\tau_\sigma)$ and $S(\sigma)$. From definition (3.8) of τ_σ and the definition of λ: for any message M having initial flit m_0 and initial arc e_0, we have that $1 \leq \tau_\sigma(m_0, e_0) \leq S(\sigma)$. If we combine this range bound with equation (2.4), and we recall the definition of the parameter $Q(\mathcal{M})$, we readily obtain the range bound

$$1 \leq \tau_\sigma(m,e) \leq S(\sigma) + Q(\mathcal{M}) - 1$$

which yields the sought inequality (3.9) when combined with the definition of $T(\tau_\sigma)$. □

The VO-transformation and Theorem 3.2 admit an intuitively appealing geometric interpretation on an n-node ULA, which is depicted schematically in Figure 4. Call the region of the virtual schedule plane at or below time $S = S(\sigma)$ and extending rightward $n-1$ units the *problem region*. We partition the problem region into disjoint *base regions*. The ith base region, for each $i \geq 0$, is a parallelogram (slanting from southeast to northwest) whose corners are the points with coordinates $(Si, 1)$, $(S(i+1)-1, 1)$, $(S(i-1)+1, S)$ and (Si, S). (Note that the only relevant portion of the 0th base region is its intersection with the positive quadrant). Each message M in the chatting problem is assigned a *home region*, namely, that base region that contains the southwest corner of the rectangle that represents M's (virtual) journey through the ULA; this corner is the point $(x, \sigma(m_0, \eta_x))$, where $\eta_x = e_0 = (x, x+1)$. Superimposed on this structure, each base region R is associated to the right with an *extended region* that is the trapezoid of corners $(S(i+1)-1, 1)$, (Si, S), $(N-2, 1)$, and $(N-2, S)$. (Note that the ith extended region is contained in all the previous $i-1$ ones).

The chatting schedule τ is obtained from the virtual chatting schedule σ by applying the transformation specified in (3.8). One can view this transformation as separate applications of the LS transformation to each pair consisting of a base region and its

associated extended region, *with respect to suitably translated axes.* More precisely, for the ith base region-extended region pair, if we set level$(\eta_{Si+j}) = j$, for some j in the range $-(S-1) \leq j \leq S-1$ (that is, if we put the origin of our new axes at point $(Si, 0)$) and then apply transformation (3.6) to all messages whose home region is the ith base region, then we obtain exactly schedule (3.8).

Note that any message M originating in the first half of its home region has now level$(e_0) < 0$, hence will have an earlier departure time in the chatting schedule. Indeed, note that in Figure 4(a), the base regions in the virtual schedule plane slant "downwards" as we move along the positive direction of the x-axis, while their images in the chatting schedule plane of Figure 4(b)) slant "upwards".

Finally, it is interesting to observe that while all the extended regions in the virtual plane overlap, their images in the chatting plane are mutually disjoint. As a consequence, one might even consider an unfeasible virtual schedule $\bar{\sigma}$ with rectangles overlapping only when they belong to different home regions. Then, Theorem 3.2 would transform $\bar{\sigma}$ into a feasible chatting schedule.

The OV Transformation. The second transformation derives virtual chatting schedules from ordinary ones. The derived virtual chatting schedules are "slower" than their originating chatting schedules by no more than the length of the longest message in the chatting problem. While the endeavor of converting an ordinary chatting schedule — which is really what one wants — into a virtual chatting schedule seems somewhat unintuitive, our motivation in pursuing this transformation is that it will yield, in conjunction with Theorem 3.2, an important bound (3.14) on the quality of chatting schedules produced by the VO transformation.

Say that we are given a chatting schedule τ, of duration $T(\tau)$, for the chatting problem $\mathcal{M}$ on the FL-network G. Define the *OV-derived pseudo-schedule* σ_τ as follows.[6] In order to evaluate $\sigma_\tau(m, e)$ for a given flit m belonging to some message $M \in \mathcal{M}$ and a given arc e of G, one looks at the first flit, call it m_0, of message M and at the first arc, call it e_0, of the designated path of message M. One then assigns

$$\sigma_\tau(m,e) \stackrel{\text{def}}{=} \begin{cases} \tau(m,e_0) - (\text{level}(e_0) \bmod T(\tau)) & \text{if } \tau(m_0,e_0) > \text{level}(e_0) \bmod T(\tau) \\ \tau(m,e_0) - (\text{level}(e_0) \bmod T(\tau)) + T(\tau) & \text{otherwise.} \end{cases} \tag{3.10}$$

Theorem 3.3 (OV-derived virtual chatting schedules)
Let τ be a chatting schedule of duration $T(\tau)$, for the chatting problem $\mathcal{M}$ on the FL-

[6] Of course, σ_τ is a virtual chatting schedule, not just a pseudo-schedule; but this must be verified before we can legitimately assert it.

network G; let σ_τ be the pseudo-schedule that is OV-derived from τ. Then σ_τ is a virtual chatting schedule for $\mathcal{M}$ of duration

$$S(\sigma_\tau) < T(\tau) + L(\mathcal{M}). \tag{3.11}$$

Proof: With no loss of generality, assume that $1 \leq \tau(m,e) \leq T(\tau)$, for all flits m belonging to the chatting problem $\mathcal{M}$ and all arcs e on designated paths of the host network G.

We must first establish that σ_τ is a virtual chatting schedule for $\mathcal{M}$. Since τ is a chatting schedule, hence enjoys both the bufferless-nodes and message-integrity properties ((2.2) and (2.3), respectively), one readily establishes the message transmission and integrity properties (3.2) for σ_τ. Let us, therefore, focus on verifying that σ_τ is free from arc conflicts. Suppose, for contradiction, that there exist flits m and m' and an arc e for which $\sigma_\tau(m,e) = \sigma_\tau(m',e)$. For the sake of generality, say that these two occurrences of arc e are associated with distinct designated paths, having, respectively, initial arcs e_0 and e_0'. From definition (3.10), then, we infer

$$(\tau(m,e_0) - \tau(m',e_0')) + ((\text{level}(e_0') - \text{level}(e_0)) \bmod T(\tau)) = \alpha T(\tau), \tag{3.12}$$

where $\alpha \in \{-1,0,1\}$. Now, since τ enjoys the bufferless-nodes property (2.2), we have

$$\tau(m,e_0) = \tau(m,e) + \text{level}(e_0) - \text{level}(e),$$

and

$$\tau(m',e_0') = \tau(m',e) + \text{level}(e_0') - \text{level}(e).$$

Using these relations in relation (3.12) yields

$$(\tau(m,e) - \tau(m',e)) + (\delta(e_0) - \delta(e_0')) = \alpha T(\tau), \tag{3.13}$$

where, for all arcs e'', $\delta(e'') \stackrel{\text{def}}{=} \text{level}(e'') - (\text{level}(e'') \bmod T(\tau))$. If we now reduce both sides of equation (3.13) modulo $T(\tau)$ and observe that, for any arc e'', $\delta(e'') \bmod T(\tau) = 0$, we obtain the equation

$$\tau(m,e) \bmod T(\tau) = \tau(m',e) \bmod T(\tau).$$

Given our assumption that $1 \leq \tau(m,e), \tau(m',e) \leq T(\tau)$, this equation yields

$$\tau(m,e) = \tau(m',e),$$

whence, as τ is arc-conflict free, $m = m'$. This verifies that σ_τ is arc-conflict free, hence is a virtual schedule.

We turn now to bounding the duration of σ_τ. By definition (3.10), for any message M whose first flit is m_0 we have $1 \leq \sigma_\tau(m_0, e_0) \leq T(\tau)$. By the virtual message integrity property (3.4), then, for all flits of M,

$$1 \leq \sigma_\tau(m, e) \leq T(\tau) + L(\mathcal{M}) - 1.$$

This range inequality yields the sought bound (3.11), when combined with the definition of $S(\sigma)$. □

Considerations similar to those following Theorem 3.2 can be made to develop a graphical view of the OV-transformation (3.10) on a ULA, as illustrated in Figure 5.

Theorems 3.2 and 3.3 yield the following relation between the duration $T_{\text{opt}}(\mathcal{M})$ of the optimal ordinary chatting schedule τ_{opt} for a given chatting problem $\mathcal{M}$ and the duration $T_{\text{opt}}^{(\text{VO})}(\mathcal{M})$ of the VO-derived chatting schedule $\tau_{\sigma_{\text{opt}}}$ that is derived from the optimal virtual chatting schedule σ_{opt} for problem $\mathcal{M}$. The reader should keep in mind that all uses of the word "optimal" here are within the context of our computing model, as spelled out in Section 2.

Corollary 3.1

$$T_{\text{opt}}^{(\text{VO})}(\mathcal{M}) < 2T_{\text{opt}}(\mathcal{M}) + L(\mathcal{M}) - 1. \tag{3.14}$$

Proof: By Theorem 3.2, $T_{\text{opt}}^{(\text{VO})}(\mathcal{M}) < S(\sigma_{\text{opt}}) + Q(\mathcal{M})$. By Theorem 3.3, $S(\sigma_{\text{opt}}) \leq T_{\text{opt}}(\mathcal{M}) + L(\mathcal{M}) - 1$. By Proposition 2.1, $Q(\mathcal{M}) \leq T_{\text{opt}}(\mathcal{M})$. □

4 Virtual Schedules via Rectangle Compaction

We now develop a strategy for producing chatting schedules for ULAs. Our approach first derives a virtual chatting schedule σ for a given chatting problem $\mathcal{M}$ and then employs the VO transformation (3.8) to obtain the desired chatting schedule. The virtual schedule σ is obtained by solving the rectangle compaction problem that corresponds to problem $\mathcal{M}$, as in Section 3.3. Using known results about a problem that is equivalent to rectangle compaction, we obtain, in linear time, a virtual chatting schedule that is within a small constant factor of optimal. The quality of the derived chatting schedule can then be estimated via the bound (3.14).

4.1 Unit-Length Messages

Let us consider first the special class of chatting problems $\mathcal{M}$ wherein all messages in $\mathcal{M}$ have unit length, so $L(\mathcal{M}) = 1$. In the geometric representation of virtual chatting

schedules for $\mathcal{M}$ (Section 3.3), any addressed message $M \in \mathcal{M}$ that originates at PE u of the host network and is destined for PE v corresponds to a horizontal segment s_M whose projection on the x-axis is the closed interval $(s_M{\downarrow}x) \stackrel{\text{def}}{=} [\eta_u, \eta_{v-1}]$. The rectangle compaction problem, which characterizes the general virtual chatting scheduling problem, thus reduces in this case to the problem of arranging the segments s_M on the minimum number of *tracks* (that is, lines parallel to the x-axis at integer coordinates) so that no two segments overlap. This problem admits an efficient optimal solution. To wit:

Theorem 4.4 *The rectangle compaction problem that represents a chatting problem $\mathcal{M}$ with $L(\mathcal{M}) = 1$ can be solved in (low-degree) polynomial time. In the optimal solution, the segments are allocated to precisely $C(\mathcal{M})$ tracks.*

Proof: We prove the theorem by reducing the compaction problem for line-segments to the problem of node-coloring interval graphs.

Consider the *undirected* graph, call it Γ, whose nodes are the messages in the chatting problem $\mathcal{M}$ and whose edges comprise all pairs of messages $\{M', M''\}$ from $\mathcal{M}$ for which the line-segments $(s_{M'}{\downarrow}x)$ and $(s_{M''}{\downarrow}x)$ overlap. Note that Γ is the *interval graph* [6] that is induced by the set of line-segments $\{(s_M{\downarrow}x) \ : \ M \in \mathcal{M}\}$. If two segments $(s_{M'}{\downarrow}x)$ and $(s_{M''}{\downarrow}x)$ cannot be allocated to the same track, then the pair $\{M', M''\}$ is an edge of Γ. If we associate tracks with colors, then the compaction problem is equivalent to finding a minimum coloring of the nodes of Γ such that no two adjacent nodes are given the same color. All segments that correspond to like-colored nodes will be allocated to the same track.

The minimum node-coloring problem for interval graphs admits a simple (low-degree) polynomial-time solution. One orders the nodes according to the left endpoints of their corresponding intervals and then applies a first-fit on-line coloring strategy to the sorted sequence [6]. The number of colors used is the maximum clique size of Γ which is clearly equal, in our case, to the message congestion $C(\mathcal{M})$. □

By applying Theorem 3.2 to the virtual chatting schedule obtained by solving the rectangle compaction problem in the proof of Theorem 4.4 we obtain:

Corollary 4.2 *In (low-degree) polynomial time, one can find, for any chatting problem $\mathcal{M}$ with unit-length messages, a chatting schedule τ of duration $T(\tau) < C(\mathcal{M}) + Q(\mathcal{M}) \leq 2T_{\text{opt}}(\mathcal{M})$.*

The bound on the quality of τ comes from Proposition 2.1.

4.2 Arbitrary Length Messages

When a chatting problem has arbitrary message lengths, the reduction of optimal virtual scheduling to optimal rectangle compaction still affords one an efficient avenue to efficient chatting schedules, but not as easily as in the unit-length case.

The decision version of the general rectangle compaction problem can be easily shown to be NP-complete, being equivalent to the following NP-complete decision problem (cf. [6]):

DYNAMIC STORAGE ALLOCATION (DSA)
Instance: Set $\mathcal{A}$ of items to be stored, each $a \in \mathcal{A}$ having an integer size $s(a)$ and a *duration span* $[r(a), d(a)]$; a storage size D.
Question: Is there an *allocation schedule* for $\mathcal{A}$, that is, a function $\sigma : \mathcal{A} \rightarrow \{1, 2, \ldots, D\}$, such that, for every $a \in \mathcal{A}$, the *storage interval* $l(a) = [\sigma(a), \sigma(a) + s(a) - 1]$ is contained in $[1, D]$ and, for any $a', a'' \in \mathcal{A}$ their storage intervals overlap only if their duration spans are disjoint?

One can immediately discern a bijection between instances of the decision version of the rectangle compaction problem and instances of the DSA problem: each item $a \in \mathcal{A}$ maps onto a rectangle of height $s(a)$ and fixed projection $[r(a), d(a)]$ on the x-axis. The allocation schedule for $\mathcal{A}$ then corresponds to the desired final (compacted) placement for the rectangles. Finally, the storage size D corresponds to the difference between the maximum and minimum ordinate of any of the points of the rectangles.

Kierstead [7] proposes a polynomial approximation algorithm for the DSA problem that yields an allocation schedule which is within a factor of 6 of optimal. Described in terms of the rectangle compaction problem, Kierstead's approximation algorithm performs the following steps.

1. It increases the height of each rectangle to a power of two.
2. It orders the rectangles in nonincreasing order of height. It then replaces each rectangle of height h and projection $[a, b]$ on the x-axis by h copies of the interval $[a, b]$ in the resulting list.
3. It node-colors the interval graph induced by the above list of intervals, using a particular on-line strategy.

It is shown in [7] that, if consecutive tracks are associated with consecutive colors (representing both track numbers and colors by integers), then the h intervals generated by a rectangle of height h are allocated to a set of h consecutive tracks; hence, the algorithm

yields a feasible placement. The analysis of the competitive ratio of the on-line strategy proves that the obtained solution employs at most 6 times the number of tracks required by the optimal solution. We summarize the above discussion in the following theorem.

Theorem 4.5 ([7]) (a) *The rectangle compaction problem for a chatting problem with arbitrary-length messages is NP-hard.*
(b) *Given a chatting problem $\mathcal{M}$, there exists a polynomial-time approximation algorithm which yields a placement of the rectangles on at most $6C(\mathcal{M})$ tracks.*

By applying Theorem 3.2 to the virtual chatting schedule obtained by solving the rectangle compaction problem with the approximation algorithm given above, we obtain:

Corollary 4.3 *In (low-degree) polynomial time, one can find, for any chatting problem $\mathcal{M}$, a chatting schedule τ of duration $T(\tau) < 6C(\mathcal{M}) + Q(\mathcal{M}) \leq 7T_{\text{opt}}(\mathcal{M})$.*

Once again, the bound on the quality of the chatting schedule τ derives from Proposition 2.1.

5 Virtual Schedules via Problem Decomposition

This section is devoted to a technique for deriving moderately efficient virtual chatting schedules for chatting problems by decomposing each problem $\mathcal{M}$ into (roughly) $\log L(\mathcal{M})$ problems on the basis of message length. The strategy produces a virtual chatting schedule for a given chatting problem $\mathcal{M}$ that is within a factor of $O(\log L(\mathcal{M}))$ of optimal. The observation that enables this strategy is that, when one is presented with a chatting problem $\mathcal{M}$ all of whose messages have the same length, one can derive an optimal virtual chatting schedule for $\mathcal{M}$ without knowing the specific (common) length of its messages. We then apply this strategy to two-dimensional mesh networks. Of course, in order to obtain the real chatting schedules that one ultimately seeks, one must apply the VO-transformation (3.8) to the derived virtual chatting schedule.

5.1 Decomposing Problems by Message Length

We develop our strategy in two steps. First, we show that the problem of finding an optimal virtual chatting schedule for messages of any common length is computationally equivalent to the problem of finding an optimal virtual chatting schedule for messages of unit length. We then parlay this observation into a strategy for converting an efficient virtual chatting schedule for unit-length messages into a virtual chatting schedule for an arbitrary chatting problem $\mathcal{M}$, that is at worst a factor of $O(\log L(\mathcal{M}))$ less efficient.

In order to formalize our strategy, let us focus on an FL-network G and an indexed family $\mathbf{M} = \{\mathcal{M}_\ell\}_{\ell \in N}$ of *related, uniform* chatting problems for G. These problems are *uniform* in the sense that each problem in $\mathbf{M}$ consists of like-length addressed messages; specifically, say that, for $\ell = 1, 2, \ldots$, the ℓth problem $\mathcal{M}_\ell \in \mathbf{M}$ is a set of addressed messages of length ℓ. The problems in $\mathbf{M}$ are *related* in the sense that, for all pairs of nodes u, v of G, there is a message from u to v in one of the problems $\mathcal{M}_\ell \in \mathbf{M}$ *if, and only if*, there is a message from u to v in each of the problems in $\mathbf{M}$. The following theorem shows that finding a virtual chatting schedule for any one $\mathcal{M}_\ell \in \mathbf{M}$ is essentially equivalent to finding a virtual schedule for every problem in $\mathbf{M}$.

Theorem 5.1 *Given the family $\mathbf{M} = \{\mathcal{M}_\ell\}_{\ell \in N}$ of related, uniform chatting problems on the FL-network G:*
(a) *One can transform any virtual chatting schedule σ_1 for the problem $\mathcal{M}_1$ into a virtual chatting schedule σ_ℓ for the problem $\mathcal{M}_\ell$ of duration $S(\sigma_\ell) = \ell S(\sigma_1)$.*
(b) *One can transform any virtual chatting schedule σ_ℓ for the problem $\mathcal{M}_\ell$ into a virtual chatting schedule σ_1 for the problem $\mathcal{M}_1$ of duration $S(\sigma_1) = \lfloor S(\sigma_\ell)/\ell \rfloor$.*
(c) *The durations $S_{\text{opt}}(\mathcal{M}_1)$ and $S_{\text{opt}}(\mathcal{M}_\ell)$ of the optimal virtual chatting schedules for problems $\mathcal{M}_1$ and $\mathcal{M}_\ell$, respectively, stand in the relation*

$$S_{\text{opt}}(\mathcal{M}_\ell) = \ell S_{\text{opt}}(\mathcal{M}_1).$$

Proof: **(a)** Say, for clerical simplicity, that the virtual chatting schedule σ_1 for the problem $\mathcal{M}_1$ is normalized so that $1 \leq \sigma_1(m, e) \leq S(\sigma_1)$ for all appropriate m and e. Define the function σ_ℓ as follows. Let $M = \langle m_0 \rangle$ and $M' = \langle m'_0, \ldots, m'_{\ell-1} \rangle$ be addressed messages in problems $\mathcal{M}_1$ and $\mathcal{M}_\ell$, respectively, that share a source and destination. Then, for all flits m'_h of M' and all arcs e along the designated path from the source to the destination of M' (hence, also, of M):

$$\sigma_\ell(m'_h, e) = \ell(\sigma_1(m_0, e) - 1) + 1 + h.$$

It is a straightforward exercise to verify that σ_ℓ is a virtual chatting schedule for $\mathcal{M}_\ell$ with duration $S(\sigma_\ell) = \ell S(\sigma_1)$.

(b) Using the same notation as in part (a), let σ_ℓ be normalized so that $1 \leq \sigma_\ell(m, e) \leq S(\sigma_\ell)$ for all appropriate m and e. Observe that the virtual message integrity property (3.4) guarantees that, for message M', the sequence of time steps $\langle \sigma_\ell(m'_0, e), \sigma_\ell(m'_1, e), \ldots, \sigma_\ell(m'_{\ell-1}, e) \rangle$ is a sequence of consecutive integers in the set $\{1, 2, \ldots, S(\sigma_\ell)\}$. Since message M' has length ℓ, there must, therefore, be a unique index $0 \leq j_\ell \leq S(\sigma_\ell) - 1$ such that $\sigma_\ell(m'_j, e) \bmod \ell = 0$. It follows that the mapping

$$\sigma_1(m_0, e) = \frac{1}{\ell} \sigma_\ell(m_{j_\ell}, e)$$

is a function. In fact, one verifies easily that σ_1 is a virtual chatting schedule for $\mathcal{M}_1$ of duration $S(\sigma_1) = \lfloor S(\sigma_\ell)/\ell \rfloor$.

(c) From Part (a), we have $S_{\text{opt},\ell} \leq \ell S_{\text{opt},1}$. From Part (b), we have $S_{\text{opt},1} \leq \lfloor S_{\text{opt},\ell}/\ell \rfloor$. If either of these relations were not an equality, it would follow that $S_{\text{opt},\ell} < \ell \lfloor S_{\text{opt},\ell}/\ell \rfloor$, which is absurd. □

We now build on the single-message-length strategy implicit in Theorem 5.1(a) to devise a strategy for arbitrary chatting problems. Roughly speaking, this general strategy partitions an arbitrary problem into a sequence of single-message-length problems which are then solved sequentially. Although the virtual chatting schedules produced by the general strategy cannot deviate from optimality by more than a factor of $\log L(\mathcal{M})$, these schedules probably do deviate by that much in general.

In the sequel, denote by $S_{\text{opt}}(\mathcal{M})$ the duration of the optimal virtual chatting schedule for the chatting problem $\mathcal{M}$.

Theorem 5.2 *Let us be given an FL-network G and a procedure P that produces, for any uniform chatting problem $\mathcal{M}'$ on G, a virtual chatting schedule $\sigma_{\mathcal{M}'}$ whose duration is within the factor α of $S_{\text{opt}}(\mathcal{M}')$. One can transform procedure P into a procedure P' that produces,* for any chatting problem $\mathcal{M}$ for G, *a virtual chatting schedule $\sigma_{\mathcal{M}}$ of duration*

$$S(\sigma_{\mathcal{M}}) \leq 2\alpha(\lceil \log L(\mathcal{M}) \rceil + 1) S_{\text{opt}}(\mathcal{M}). \tag{5.1}$$

Proof: We begin by partitioning a given chatting problem $\mathcal{M}$ into subproblems all of whose messages have (roughly) the same length. Specifically, we partition $\mathcal{M}$ into the subproblems:

$$\mathcal{M} = \bigcup_{i=0}^{\lceil \log L \rceil} \mathcal{M}_i,$$

where each subproblem $\mathcal{M}_i$ contains exactly those messages of $\mathcal{M}$ whose length ℓ lies in the range $2^{i-1} < \ell \leq 2^i$.

We turn now to the second step in our strategy. For each integer l, let $\mathcal{M}_{i,l}$ be the set of addressed messages obtained by replacing each addressed message of $\mathcal{M}_i$ with an l-flit message having the same source and destination. We obtain a virtual chatting schedule for subproblem $\mathcal{M}_i$ of $\mathcal{M}$ via the following 3-step procedure.

1. Use procedure P (as described in the statement of the Theorem) to derive a virtual chatting schedule $\sigma_{i,1}$ for problem $\mathcal{M}_{i,1}$.

2. Use Theorem 5.1(a) to convert $\sigma_{i,1}$ to a virtual chatting schedule $\sigma_{i,2^i}$ for problem $\mathcal{M}_{i,2^i}$.

3. Adapt $\sigma_{i,2^i}$ to obtain a virtual chatting schedule σ_i for subproblem $\mathcal{M}_i$ as follows. A message of length ℓ in $\mathcal{M}_i$ is scheduled by σ_i in just the way that $\sigma_{i,2^i}$ would schedule it, except that the last $2^i - \ell$ flits of the message are ignored.

Thus, we have:

$$\begin{array}{lll} S(\sigma_i) & = S(\sigma_{i,2^i}) & \text{by design} \\ & = 2^i S(\sigma_{i,1}) & \text{by Theorem 5.1(a)} \\ & \leq \alpha 2^i S_{\text{opt}}(\mathcal{M}_{i,1}) & \text{by hypothesis} \\ & = \alpha S_{\text{opt}}(\mathcal{M}_{i,2^i}) & \text{by Theorem 5.1(c)} \\ & \leq 2\alpha S_{\text{opt}}(\mathcal{M}) & \end{array}$$

The last inequality follows from the fact that $\mathcal{M}_{i,2^i}$ can be considered for scheduling purposes as a subset of $\mathcal{M}$ where some messages have been increased in length by a factor smaller than 2.

Finally, we obtain the desired virtual chatting schedule $\sigma_{\mathcal{M}}$ for $\mathcal{M}$ by consecutively scheduling the subproblems $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_{\lceil \log L \rceil}$ using the schedules $\sigma_0, \sigma_1, \ldots, \sigma_{\lceil \log L \rceil}$. Obviously, the duration of schedule $\sigma_{\mathcal{M}}$ satisfies inequality (5.1). □

5.2 Applications to Two-Dimensional Meshes

The strategy of Theorem 5.2 can be applied only when one has access to the posited Procedure P that produces efficient virtual chatting schedules for uniform chatting problems. By Theorem 5.1, it suffices to have a Procedure P that produces efficient virtual chatting schedules for chatting problems that contain only unit-length messages.

As a simple illustration of the strategy of problem decomposition by message length, consider chatting problems on ULAs. Theorem 4.4 supplies us with an efficient procedure that constructs, for any one-flit-message chatting problem $\mathcal{M}$ on a ULA, a virtual chatting schedule σ with optimal duration $S(\sigma) = C(\mathcal{M})$. Using this procedure as the Procedure P in Theorem 5.2, we obtain, for an arbitrary chatting problem $\mathcal{M}'$ on a ULA, a virtual chatting schedule σ' of duration $S(\sigma') \leq 2\,(\lceil \log L(\mathcal{M}') \rceil + 1)\, C(\mathcal{M}')$. We should remark that, in this case, the strategy of Section 4 which simultaneously deals with messages of all lengths, will generally yield a virtual chatting schedule with a smaller duration.

As the preceding illustration suggests, finding an efficient schedule-generating procedure, even for one-flit-message chatting problems, seems to require network-specific considerations. We have succeeded in finding such a procedure for mesh networks.

Since our overall strategy of deriving chatting schedules from virtual chatting sched-

ules is predicated on chatting within a host FL-network (so that the derived schedule has predictable efficiency), we concentrate here on a somewhat impoverished version of the standard mesh network.

The $N \times N$ *Eastward-Southward mesh network* (*ESM*, for short) has node-set $V = \{(i,j) : 0 \leq i,j \leq N-1\}$ and arcs connecting each node (i,j) to node $(i+1,j)$ providing that $i < N-1$, and to node $(i,j+1)$, providing that $j < N-1$. One verifies easily that the $N \times N$ ESM is an FL-network: for $\ell = 0,1,\ldots,2N-2$, the ℓth level comprises the set $V_\ell = \{(i,j) : i+j = \ell\}$. In conformance with the metaphor implicit in the name of ESMs, we say that the rows of an ESM run eastward, while the columns run southward.

Just as ULAs are useful intermediate structures for handling chatting problems on linear arrays (Section 3.4), ESMs are useful for handling chatting problems on "full meshes" — whose arc sets connect each node (i,j) to nodes $(i \pm 1, j)$ and $(i, j \pm 1)$, providing that the addition/subtraction keeps one within the node-set. Specifically, a "full mesh" can be obtained by superposing four appropriately rotated ESMs.

Our study presumes the existence of designated paths within the host FL-network. To this end, for each potential source-destination pair (i,j), (i',j') in an ESM (note that $i \leq i'$ and $j \leq j'$), we designate the "one-turn" path that proceeds eastward along *row* i from node (i,j) to node (i',j), then "turns the corner" to proceed thence southward along *column* j to node (i',j').

We turn now to the task of finding efficient virtual chatting schedules for chatting problems all of whose messages have unit length. Our scheduling algorithms has the following overall structure. We start at node $(N-1,0)$, which is the southwest corner of the ESM, and we scan its nodes along the diagonals given by

$$\{(i,j) : i - j = \textit{constant}\};$$

along each diagonal, we scan nodes in increasing order of column index. We process each message when we encounter the node where it "turns" from the row-traversing part of its path to the column-traversing part; messages that share a corner node are ordered arbitrarily. We assign each message the earliest virtual time slot that has not already been assigned.

To flesh out the description of the algorithm, we need only specify how we identify the earliest available virtual time slot. For this purpose, we maintain the following auxiliary data structures. Let $\mathcal{M}$ be the chatting problem being scheduled. For each row-index i and each column-index j, where $i,j \in \{0,1,\ldots,N-1\}$, we maintain arrays $\text{row}_i[s]$ and $\text{col}_j[s]$, each having $2C(\mathcal{M})$ entries. We initialize each entry of each row array to 0 and each entry of each column array to $N-1$. The purpose of these arrays is to maintain the following information, which is updated in the course of the scanning process. Say that,

in the course of scanning the ESM, we find ourselves at node (i,j), processing message M, which "turns the corner" at that node. Then, for each virtual time s:

- $\text{row}_i[s] = k$ just when node (i,k) is the eastmost "turning node" in row i of any message that has thus far been assigned virtual time slot s; this means that only messages beginning at nodes (i,m), where $m \geq k$, can be scheduled along row i in virtual time slot s.
- $\text{col}_j[s] = h$ just when node (h,j) is the northmost "turning node" in column j of any message that has thus far been assigned virtual time slot s; this means that only messages destined for nodes (m,j), where $m \leq h$, can be scheduled along column j in virtual time slot s.

Now, say that the scheduling algorithm is scanning "turning node" (i,j) and is processing some message — call it M_0 — whose source is node (i,k) and whose destination is node (h,j): perforce, $h > i$, and $k < j$. (Of course, the algorithm could have to process several messages at each "turning node".) The algorithm seeks the smallest available virtual time slot, by probing, in turn, for $s = 1, 2, \ldots, 2C(\mathcal{M})$, the pair of array entries $\text{row}_i[s]$ and $\text{col}_j[s]$. Virtual time slot s_0 is available just when $\text{row}_i[s_0] \leq k$ and $\text{col}_j[s_0] > h$. When this compound condition holds, message M is assigned virtual time slot s_0, and the array entries are updated to reflect this: $\text{row}_i[s_0]$ is reset to j, and $\text{col}_j[s_0]$ is reset to i.

To validate this algorithm, we must argue that there is always an unassigned virtual time slot available in the two arrays. Assume that this were not the case; in particular, say that message M_0 of the preceding paragraph cannot be scheduled. It follows that at each possible virtual time slot, either some message is using the arc $\langle (i,k)(i,k+1) \rangle$ or the arc $\langle (h-1,j),(h,j)$. But this means that at least one of these arcs is traversed *to this point* by at least $C(\mathcal{M})$ messages, *in addition to message* M_0. This is absurd, by definition of arc-congestion. Building on this observation, it is easy to verify that our algorithm produces a legitimate virtual chatting schedule for problem $\mathcal{M}$. Thus, we have established the following theorem.

Theorem 5.3 *Let $\mathcal{M}$ be a chatting problem with single-flit messages on the $N \times N$ ESM. In time $O(|\mathcal{M}|C(\mathcal{M}))$, one can produce a virtual chatting schedule σ for $\mathcal{M}$, of duration $S(\sigma) \leq 2C(\mathcal{M})$.*

One can now invoke Theorems 5.2 and 3.2 to obtain the following result.

Corollary 5.1 *Let $\mathcal{M}$ be a chatting problem on the $N \times N$ ESM. In time $O(|\mathcal{M}|C(\mathcal{M}))$, one can produce a chatting schedule τ for $\mathcal{M}$, of duration*

$$T(\tau) \leq 4\left(\lceil \log L(\mathcal{M}) \rceil + 1\right) C(\mathcal{M}) + L(\mathcal{M}) + 2N.$$

6 Virtual Schedules via Network Cuts

The scheduling strategies of Section 5 and (less obviously) of Section 4 make crucial use of the lengths of the messages in a chatting problem. In contrast, the strategy of this final section produces chatting schedules by decomposing the given chatting problem with respect to a suitable set of *cuts* of the host FL-network G.

6.1 Chatting Schedules via Network Decomposition

A *cutset* of the network G is a set A of arcs whose removal partitions G into subnetworks $G_A^{(1)}$ and $G_A^{(2)}$ that are *disjoint* in the following sense: every path in G that originates at a node of $G_A^{(1)}$ and terminates at a node of $G_A^{(2)}$, or vice versa, crosses some arc in A. One can derive a (virtual) chatting schedule for a chatting problem $\mathcal{M}$ for G, via the following recursive strategy that successively decomposes G via cuts.

1. Determine a cutset A of network G, and partition G into subnetworks $G_A^{(1)}$ and $G_A^{(2)}$ by removing the arcs of A.
2. Find a (virtual) chatting schedule for the chatting problem $\mathcal{M}_A \subseteq \mathcal{M}$ that comprises those addressed messages of $\mathcal{M}$ whose designated paths contain at least one arc of A.
3. Let $\mathcal{M}_A^{(1)} \subseteq \mathcal{M} \backslash \mathcal{M}_A$ (resp., $\mathcal{M}_A^{(2)} \subseteq \mathcal{M} \backslash \mathcal{M}_A$) be the set of addressed messages of $\mathcal{M}$ whose designated paths are totally contained in network $G_A^{(1)}$ (resp., network $G_A^{(2)}$). Derive (virtual) chatting schedules for problems $\mathcal{M}_A^{(1)}$ and $\mathcal{M}_A^{(2)}$ by recursively applying this cutting procedure to networks $G_A^{(1)}$ and $G_A^{(2)}$, respectively.

The potential value of this strategy resides in the facts that (a) the three chatting problems $\mathcal{M}_A$, $\mathcal{M}_A^{(1)}$, and $\mathcal{M}_A^{(2)}$ are no larger than $\mathcal{M}$; (b) $\mathcal{M}_A^{(1)}$ and $\mathcal{M}_A^{(2)}$ can be scheduled independently and simultaneously — that is, the messages in $\mathcal{M}_A^{(1)}$ can share (virtual) time slots with the messages in $\mathcal{M}_A^{(2)}$. The potential detractors from this value are: (a) any of these three chatting problems could actually comprise the entire problem $\mathcal{M}$; (b) even when problem (a) does not arise, the quality of the resulting chatting schedule depends upon the recursion depth of one's particular application of the strategy. Of course, the depth of the recursion is guaranteed not to be large when, at each level, the subnetworks $G_A^{(1)}$ and $G_A^{(2)}$ produced from the then-current G are roughly equal in size.

The potential drawbacks notwithstanding, the strategy is demonstrably useful for certain networks that have small cuts into large subnetworks. We illustrate the strategy

on two such networks.

6.2 Unidirectional Linear Arrays: A Running Example

For a ULA, any single arc is a cutset. It is, therefore, a straightforward matter to schedule the chatting problem $\mathcal{M}_A$ efficiently. Moreover, if we choose the cut-arc at each phase of the strategy judiciously, then we can decompose the current ULA into two (roughly) half-size ULAs. This guarantees that our strategy will recurse for only logarithmically many levels.

In more detail, we implement the strategy on an n-node ULA G by choosing the cutset $A = \{\eta_{\lfloor n/2 \rfloor - 1}\}$. The resulting subnetworks $G_A^{(1)}$ and $G_A^{(2)}$ will, then, be ULAs, one having $\lfloor n/2 \rfloor$ nodes and the other having $\lceil n/2 \rceil$ nodes. It follows that our strategy will recurse to a depth of $\lceil \log n \rceil$ levels.

Since no more than $C(\mathcal{M})$ flits traverse any arc of the ULA G, the chatting problem $\mathcal{M}_A$ comprises at most $C(\mathcal{M})$ addressed messages, hence can be scheduled to have duration $\leq C(\mathcal{M})$. Since the chatting problems $\mathcal{M}_A^{(1)}$ and $\mathcal{M}_A^{(2)}$ are scheduled during the same level of recursion, they can in fact can be merged into a single chatting problem, of congestion $\leq C(\mathcal{M})$, on the (disconnected) network $G_A^{(1)} + G_A^{(2)}$. It follows that they, too, admit a virtual chatting schedule of duration $\leq C(\mathcal{M})$, because messages in different problem have nonoverlapping designated paths. In fact, what we have just said about problems $\mathcal{M}_A^{(1)}$ and $\mathcal{M}_A^{(2)}$ obtains for the subproblems that are processed at each level of the recursion. Therefore, we can concatenate the chatting schedules for all of the levels of the recursion, to obtain a virtual chatting schedule σ for the entire problem $\mathcal{M}$ of duration $S(\sigma) \leq \lceil \log n \rceil C(\mathcal{M})$. We can now apply Theorem 3.2 to transform this virtual chatting schedule into an (ordinary) chatting schedule τ for $\mathcal{M}$, of duration $T(\tau) \leq \lceil \log n \rceil C(\mathcal{M}) + Q(\mathcal{M})$.

Of course, the techniques in Section 4 yield better schedules for chatting on ULAs. We turn, therefore, to a more complex network, where network decomposition produces the best schedules we know how to produce.

6.3 Bidirectional Trees

A *bidirectional tree* (*BT*, for short) is a complete tree wherein any two adjacent nodes u and v are connected by two opposing arcs, (u, v) and (v, u). Given any pair of nodes x and y of a BT, we designate the (unique) shortest path from x to y as the path along which all messages having source x and destination y are routed. Clearly, a BT is not an

FL-network; however, any set of designated paths that traverse a given arc of the BT do form an FL-subnetwork of the tree; hence, for such a subnetwork, one can use the VO transformation of (3.8) to transform an efficient virtual chatting schedule to an efficient ordinary chatting schedule. Limiting the utility of this observation is the fact that virtual chatting schedules for different FL-subnetworks of a BT cannot be concatenated into a single virtual chatting schedule for the entire tree, since they rely on different partitions of the nodes into levels.

As noted in Section 6.1, the efficiency of chatting schedules produced by network decomposition depends on keeping the depth of the decomposition recursion small, which, in turn, depends on one's ability to recursively cut the network into subnetworks of roughly equal sizes. The following result indicates that this is always possible for the FL-subnetworks of BTs.

Proposition 6.1 ([12]) *Every n-node BT of maximum node-degree $\delta > 1$ can be partitioned, by removing at most two arcs, into two BTs, each having at least $\lfloor n/\delta \rfloor$ nodes, and neither having more than $\lceil (\delta - 1)n/\delta \rceil$ nodes.*

Proposition 6.1 guarantees that there is always a way to recursively decompose an n-node BT of maximum node-degree $\delta > 1$ using at most $\lceil \delta \log n \rceil$ levels of recursion. Moreover, a simple inductive search allows one to find the separating arcs efficiently. This fact leads to the following chatting strategy for BTs.

Let $\mathcal{M}$ be a chatting problem on a BT $\mathcal{T}$. We schedule problem $\mathcal{M}$ on tree $\mathcal{T}$ by recursively decomposing $\mathcal{T}$ as in Proposition 6.1. At each level of the recursion, we remove at most two arcs of $\mathcal{T}$. For each removed arc, we derive a virtual chatting schedule σ of duration $S(\sigma) \leq C(\mathcal{M})$. By applying Theorem 3.2 to this collection of virtual schedules, we obtain a schedule for all the messages crossing the cut of duration $\leq 2(C(\mathcal{M})+Q(\mathcal{M}))$. Note that schedules corresponding to different cuts at the same level of recursion can be merged into a single schedule, since messages in different sets have non overlapping paths. Finally, concatenating the schedules for all levels of recursion yields a chatting schedule τ for $\mathcal{M}$ of duration $T(\tau) \leq 2(C(\mathcal{M}) + Q(\mathcal{M}))\lceil \delta \log n \rceil$. In summary we have:

Theorem 6.1 *Let $\mathcal{M}$ be a chatting problem for a BT of n nodes and maximum node-degree $\delta > 1$. One can efficiently produce a chatting schedule τ for $\mathcal{M}$, of duration $T(\tau) \leq 2(C(\mathcal{M}) + Q(\mathcal{M}))\lceil \delta \log n \rceil$.*

ACKNOWLEDGMENTS. This research was supported in part by the Istituto Trentino di Cultura, through the Leonardo Fibonacci Institute, in Trento, Italy. Further research support for S. N. Bhatt by NSF Grant CCR-88-07426, by NSF/DARPA Grant CCR-89-08285, and by Air Force Grant AFOSR-89-0382; for G. Bilardi by CNR and the Italian

Ministry of Research; for G. Pucci by the CNR project "Sistemi Informatici e Calcolo Parallelo"; for A. Ranade by AFOSR Grant F49620-87-C-0041; and for A. L. Rosenberg by NSF Grant CCR-90-13184.

We gratefully acknowledge constructive discussions with Franco P. Preparata.

References

[1] S.N. Bhatt, G. Pucci, A. Ranade, A.L. Rosenberg (1992): Scattering and Gathering Messages in Networks of Processors. *IEEE Trans. Comput.*, to appear. See also *Advanced Research in VLSI and Parallel Systems 1992* (T. Knight and J. Savage, eds.) 318-332.

[2] A. Borodin and J.E. Hopcroft (1985): Routing, merging, and sorting on parallel models of computation. *J. Comp. Syst. Sci. 30*, 130-145.

[3] W.J. Dally and C.L. Seitz (1987): Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comp., C-36*, 547-553.

[4] U. Feige and P. Raghavan (1992): Exact analysis of hot-potato routing. *33rd IEEE Symp. on Foundations of Computer Science*, 553-562.

[5] S. Felperin, P. Raghavan, E. Upfal (1992): A theory of wormhole routing in parallel computers.. *33rd IEEE Symp. on Foundations of Computer Science*, 563-572.

[6] H.A. Kierstead (1988): The linearity of first-fit coloring of interval graphs. *SIAM J. Discr. Math. 1*, 526-530.

[7] H.A. Kierstead (1991): A polynomial time approximation algorithm for Dynamic Storage Allocation. *Discr. Math. 88*, 231-237.

[8] G. Kortsarz and D. Peleg (1992): Approximation algorithms for minimum time broadcast. *Theory of Computing and Systems (ISTCS '92). Lecture Notes in Computer Science 601*, Springer-Verlag, N.Y., pp. 67-78.

[9] D.H. Lawrie and D.A. Padua (1984): Analysis of message switching with shuffle-exchanges in multiprocessors. In *Interconnection Networks*, IEEE Computer Soc. Press, N.Y.

[10] D.H. Linder and J.C. Harden (1991): An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes. *IEEE Trans. Comp. 40*, 2-12.

[11] D. Peleg and J.D. Ullman (1989): An optimal synchronizer for the hypercube. *SIAM J. Comput. 18*, 740-747.

[12] L.G. Valiant (1981): Universality considerations in VLSI circuits. *IEEE Trans. Comp., C-30*, 135-140.

[13] L.G. Valiant (1982): A scheme for fast parallel communication. *SIAM J. Comput. 11*, 350-361.

[14] L.G. Valiant (1989): Bulk-synchronous parallel computers. In *Parallel Processing and Artificial Intelligence* (M. Reeve and S.E. Zenith, eds.) Wiley, N. Y., pp. 15-22.

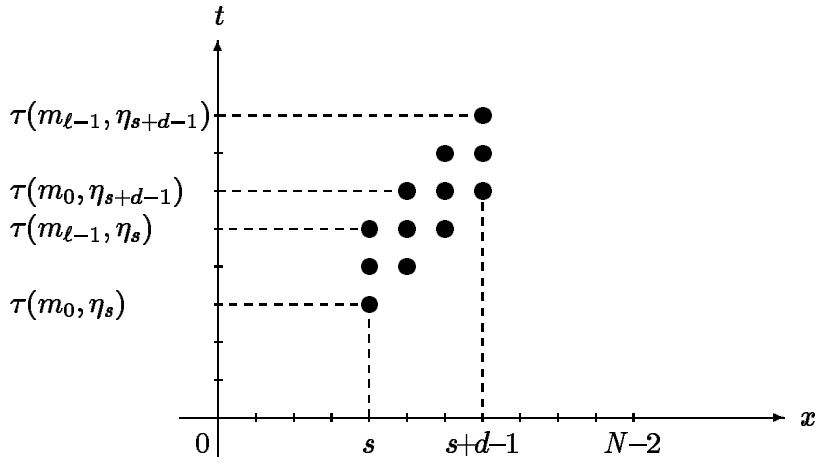


Figure 1: Graphical representation of a chatting schedule τ for one message.

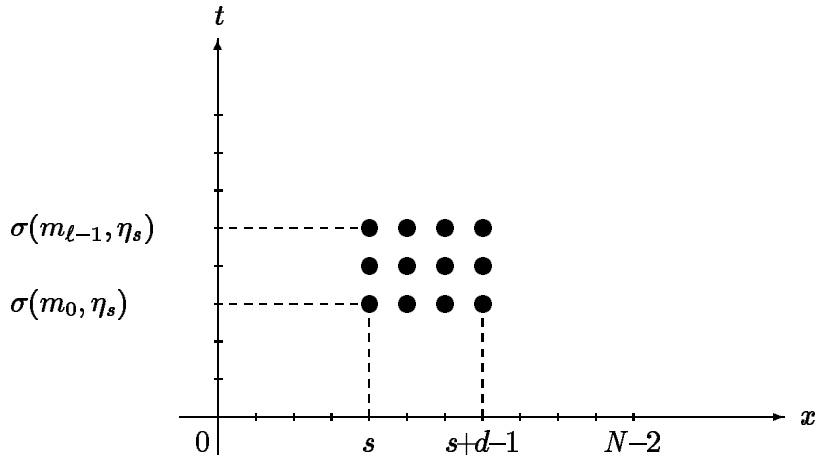


Figure 2: Graphical representation of a virtual chatting schedule σ for one message.

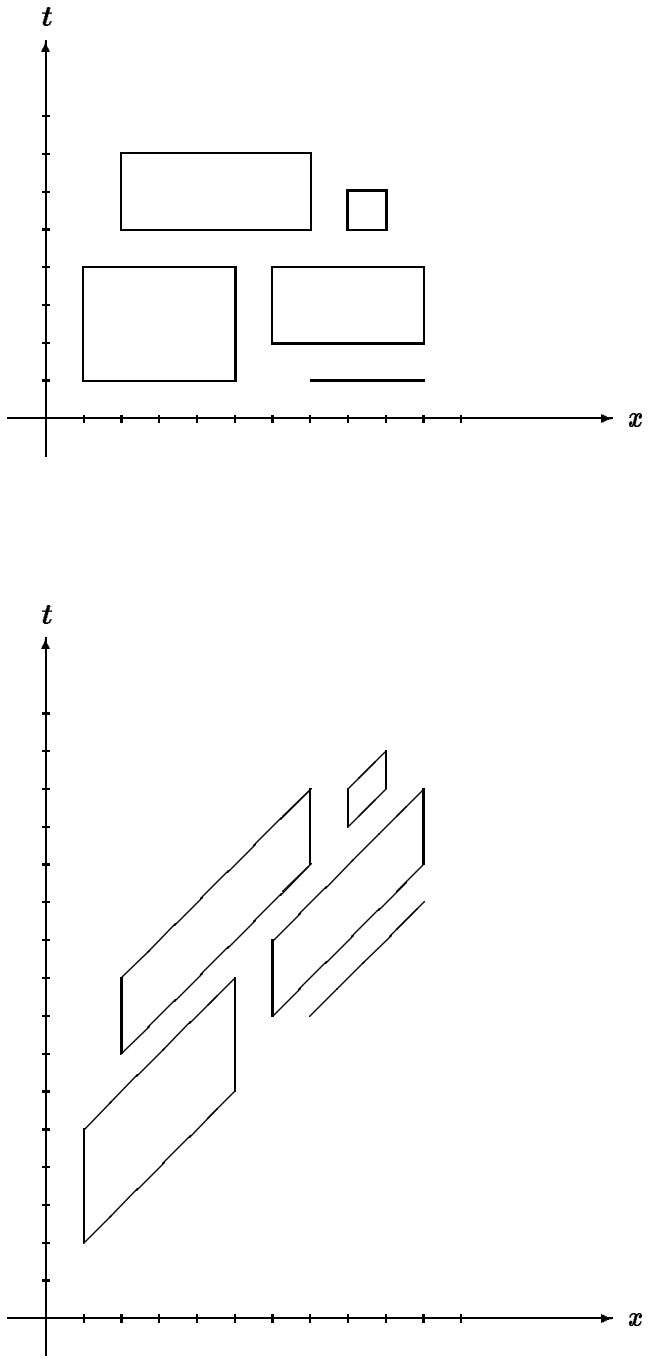
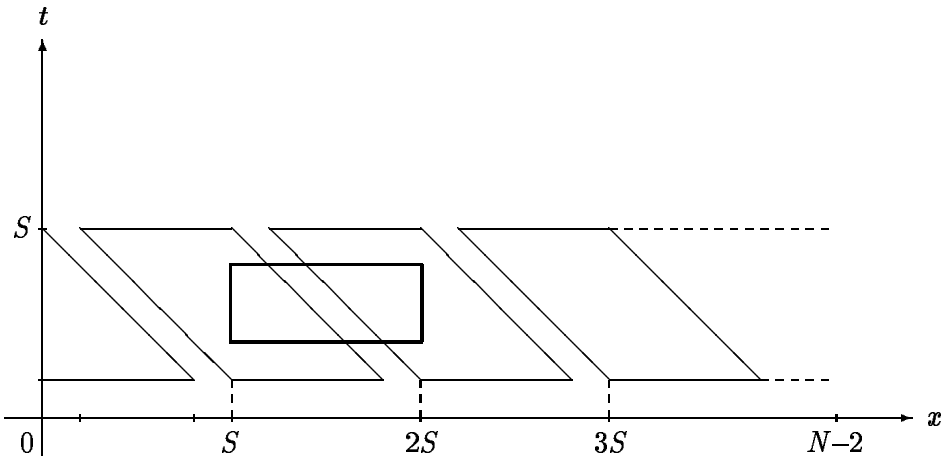
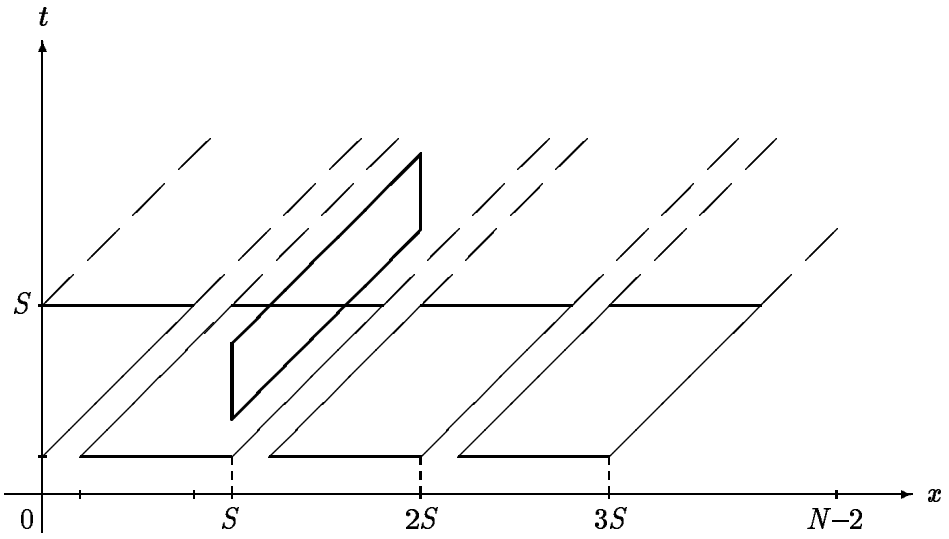


Figure 3: Graphical representation of transformation (3.6) for a ULA



(a)



(b)

Figure 4: Graphical representation of transformation (3.8) for a ULA. Continuous, dashed and thick lines denote, respectively, base regions, extended regions and messages in (a), and their images in (b).

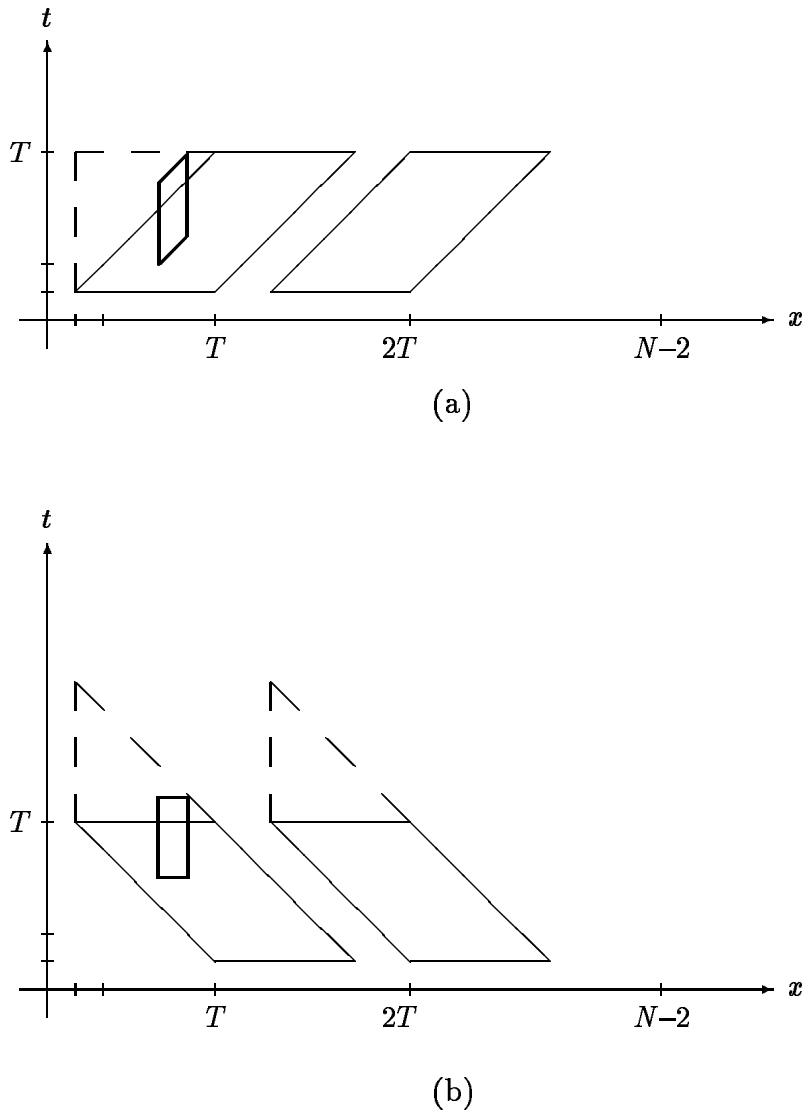


Figure 5: Graphical representation of transformation (3.10) for a ULA. Continuous, dashed and thick lines denote, respectively, base regions, extended regions and messages in (a), and their images in (b).