Mapping Concurrently-Communicating Modules onto Mesh Multicomputers Equipped with Virtual Channels

Bing-rung Tsai and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
Email: {iast,kgshin}@eecs.umich.edu

Abstract—*It is difficult to define and evaluate a meaningful performance metric when many packets are generated and exchanged concurrently in mesh-connected multicomputers equipped with wormhole switching and virtual channels. Thus, an approximate metric/cost function must be chosen so that when task modules are mapped by optimizing this function, the actual performance of the mapping is also optimized. Several low-complexity cost functions are evaluated using the simulated annealing optimization process. The mappings found by optimizing these cost functions are then fed into a flit-level simulator to evaluate their actual performance. One particular cost function is found to be very effective.*

1 Introduction

Interconnection networks equipped with wormhole switching have been widely used for contemporary multicomputers/parallel machines. In such a network, each pair of adjacent nodes is connected by a pair of unidirectional physical links/channels. A fixed number of virtual channels are time-multiplexed over each physical channel. Though most of our discussion may apply to general networks, we will focus primarily on the mesh network topology, especially k-ary 2-cubes which have been widely used in evaluating the performance of virtual-channel networks. Particularly, we will concentrate on the case where a substantial number of packets can be transmitted through the network (near) simultaneously, thus possibly causing serious traffic congestion.

Since internode communications largely depend on how communicating task modules are assigned to the nodes, our task-mapping model will consider intertask communications only, just like Model 5 in [1].

The delivery of these concurrently-transmitted packets may not be mutually independent. In many cases, task execution cannot proceed until *all* packets arrive at their destinations. Therefore, in addition to the usual latency measurement, we will use the *makespan* of a set of concurrently-sent packets for performance evaluation. The makespan of a set of packets is defined as the time span from the generation of the first packet until all the packets reach their destination.

The paper is organized as follows. Basic terms and concepts necessary for our discussion are defined in Section 2. Simulation results are presented and discussed in Section 3. The paper concludes with Section 4.

2 Preliminaries

A k-ary n-cube consists of k^n nodes arranged in an n-dimensional grid. Each node is connected to its Cartesian-coordinate neighbors in the grid. A 2-dimensional $k \times k$ flat mesh is a subgraph of a k-ary 2-cube, is not a regular graph, and has less edges than the corresponding k-ary 2-cubes (no wrap links at its boundary nodes). For convenience, we will call a k-ary 2-cube a *wrapped mesh*, or a *w-mesh* for short. Likewise, we will call a 2-dimensional flat mesh an *f-mesh*.

Flow control in a virtual-channel network is performed at three levels: routing algorithms, packet-scheduling policies, and flit-multiplexing methods. Each of these can be implemented with a variety of algorithms.

Routing: We consider only oblivious routing. A packet is routed to its destination via a fixed, shortest path. Issues related to fault-tolerance are not considered; physical and virtual channels are assumed to be fault-free. In f-meshes, e-cube routing is used. The address of each node is expressed in terms of X and Y coordi-

The work reported in this paper was supported in part by the National Science Foundation under Grant MIP-9203895, and the Office of Naval Research under Grant N00014-94-1-0229. Any opinions, findings, and recommendations in this paper are those of the authors and do not reflect the views of the funding agencies.

1063-6927/95 $4.00 © 1995 IEEE

nates. A packet is routed first in the X-direction until the Y coordinate of the node matches that of its destination node. It is then routed in the Y-direction. In w-meshes, a modified version of e-cube routing is implemented to utilize the extra communication links so that each packet is routed via a shortest path. Deadlock-freedom is ensured by using the scheme proposed in [2]. That is, the virtual channels corresponding to each unidirectional physical channel are divided into high and low channels. Routing restrictions are then imposed such that either a high channel or a low channel, but not both, is allocated to each given packet. The w-meshes need at least two virtual channels per physical channel to achieve deadlock-freedom.

Packet Scheduling: This determines which packet is allowed to access a free virtual channel in case of contention. When the number of packets to access a physical channel at the same time is larger than the number of available virtual channels, some of these packets have to be queued. So, we need to determine which packets are allowed to access the virtual channels, and which packets to be queued. When evaluating cost functions, we will mainly use the FIFO policy as a default, but other scheduling policies will also be tested when necessary.

Flit Multiplexing: This determines the way packets are time-multiplexed over a physical channel. When there are multiple virtual channels per physical channel, the packets allocated to these virtual channels are multiplexed over the physical channel. Flit multiplexing determines the order for these flits from different virtual channels to access the physical channel. In the *round-robin* (RR) multiplexing, virtual channels take turns in accessing the physical channel without using any network or packet information. RR multiplexing without any modification will henceforth be called *strict* RR. *Demand-driven* (DD) allocation can be used to remedy the waste of physical bandwidth with strict RR. With DD allocation, virtual channels will contend for use of a physical channel only if they have flits to send. Furthermore, with *CTS* (Clear-To-Send) *lookahead*, virtual channels only contend for use of a physical channel if each of them has a flit to send *and* the receiving node has room for accepting it. This can further reduce the waste of physical bandwidth. Like packet sequencing, flit multiplexing can also be priority-based, as discussed in [3].

We will use the following assumptions.

1. All mappings are one-to-one, i.e., each processor can be assigned at most one task module.
2. Accurate values of ℓ_i's (packet lengths in flits) are given.
3. A physical channel takes one unit of time to transmit a single flit. A unit of time is also called a physical-channel *cycle*.
4. There is a single-flit buffer associated with each virtual channel.
5. A packet arriving at its destination is consumed without waiting.
6. There are an even number of virtual channels associated with each physical channel in a w-mesh.

Problem Statement: Given a set of task modules and a set P of packets to be exchanged among these modules, we want to map these modules into the multicomputer so that the makespan and average latency to deliver all the packets in P may be minimized.

To select one of a large number of possible mappings, there must be a certain function to determine the quality of mapping. The most obvious choice is using the performance objective itself. In our case, the average latency of packets in the set P can be expressed as $\sum_{p_i \in P} t_i^l / \mid P \mid$, where t_i^l is the latency of packet p_i. Their makespan can be expressed as $\max_{p_i \in P}(t_i^a + t_i^l)$, where t_i^a is the generation time of p_i. In these equations, t_i^l can be expressed as $t_i^l = t_i^0 + (1/r_i)(\ell_i - 1)$. The first term, t_i^0, denotes the time span between the generation of p_i at the source node and the arrival of its header flit at the destination node. t_i^0 consists of two components: the accumulated queueing delay t_i^q and the accumulated header flit multiplexing delay t_i^x. t_i^q is the sum of queueing times at all nodes in the path waiting for an available virtual channel. t_i^x is the sum of times p_i's header flit waits at the output buffers of nodes on its path for use of physical channels. The second term, $(1/r_i)(\ell_i - 1)$, represents the time required for all other flits of p_i to arrive at the destination, which is determined by ℓ_i, which is the length of p_i, and the transmission rate, r_i, of the pipeline set up for p_i. Depending on the flit-multiplexing method used and the network condition, r_i may change with time during the transmission of p_i. Also, given a set of packets and a fixed number v of virtual channels over each physical link, t_i^q will be affected by the underlying packet-sequencing scheme. Therefore, even when the exact values of ℓ_i's are given, it is still very difficult to predict t_i^l's.

Consider the simplest case of f-meshes using strict RR multiplexing without DD allocation or CTS lookahead. We have $r_i = 1/v \ \forall i$, hence only t_i^0 needs to be calculated. Suppose v is large enough so that no packets will be blocked, then $t_i^q = 0$, and $t_i^0 = t_i^x$. However, $t_i^x \in [0, \ v * d(a_i^s, a_i^d)]$, where $d(a_i^s, a_i^d)$ denotes the Hamming distance between the source node address a_i^s and the destination address a_i^d of p_i. As v and $d(a_i^s, a_i^d)$ become larger, it is more difficult to predict t_i^x's. Furthermore, when the number of concurrent packets is large and blocking is inevitable, t_i^q is no longer 0, and the value of t_i^0 becomes even less predictable. With DD allocation or CTS lookahead, r_i is no longer a constant. It becomes even more complex in the case of w-meshes with partitioning of virtual channels for deadlock-avoidance. Thus, direct optimization of the performance objective itself is not practical. We need to come up with a certain simplified function that, when mappings are derived by optimizing it, the resulting performance is also optimized. In this paper, we will investigate low-complexity cost functions whose computational complexities are of order $\theta(|P|)$.

A packet $p_i \in P$ is characterized by its source and destination modules, ℓ_i, and t_i^a. Of these parameters, the accurate packet-generation time, t_i^a, is the most difficult to obtain beforehand, since t_i^a will be affected by the precedence relationship among modules, and is intrinsically difficult for a compiler or loader to analyze before actual execution. Even if the modules can be test-executed, packet-generation times can still vary with different executions of the same set of modules due to minor variations in the execution environment, such as clock-frequency drift. Besides, introducing time-related parameters into the optimization process can further complicate the problem by adding the scheduling into the picture. Since we are mostly interested in dealing with concurrently-communicating modules, the time window within which packets are generated, denoted by ΔT, should be relatively small. We therefore propose cost functions which ignore ΔT and assume $t_i^a = 0$, $\forall p_i \in P$. Nevertheless, as we will show in our simulations, the mappings obtained by optimizing a properly-chosen cost function still perform well when ΔT is large. Besides, the issues of packet scheduling can be handled at run-time and, as we will demonstrate, can further improve the performance of a mapping.

The following cost functions will be evaluated:

- f_1: Let $<x, y>$ denote the physical channel connecting node x and y, C_{xy} denote the maximum number of packets that share $<x, y>$, and L_i denote the set of physical channels in the path of p_i. The estimated value of makespan t_i^l, denoted as $\tilde{t}_i^l$, is computed by $z + \min\{v, \max_{<x,y>\in L_i} C_{xy}\}(\ell_i - 1)$, where z is the estimated time required for the header flit of p_i to reach its destination. z is computed as $random() * \sum_{<x,y>\in L_i} F_{xy}$, where $random()$ is a random number uniformly distributed in $(0,1)$ and F_{xy} is the total number of flits that go through $<x, y>$ during the execution of the task. The estimated makespan is computed by taking the maximum of $\tilde{t}_i^l$'s. Since r_i assumes the lowest possible value, this will be a pessimistic estimate.
- f_2: Similar to f_1, except that the estimated average value of $\tilde{t}_i^l$'s is computed.
- f_3: sum of length-distance products, i.e., the total physical bandwidth required by the packets in P. Formally, it can be expressed as $\sum_{p_i\in P} \ell_i * d(a_i^s, a_i^d)$. This cost function is shown to be quite effective in large-buffer, non-multiplexing networks [4]. However, in a virtual channel network with wormhole switching, apart from physical bandwidth, the usage of flit buffers is also a major factor in network performance.
- f_4: $\max_{0\le x,y<M} C_{xy}$, the maximum number of packets to go through a physical channel, i.e., maximum congestion.
- f_5: $\sum_{0\le x,y<M} C_{xy}$, the sum of congestion on all physical channels.
- f_6: $\max_{p_i\in P}\{\sum_{<x,y>\in L_i} F_{xy}\}$, the maximum number of flits to go through a physical channel on the paths of all $p_i \in P$.
- f_7: $\sum_{p_i\in P}\{\sum_{<x,y>\in L_i} F_{xy}\}$. Similar to f_6 but summation is taken instead. Note that f_7 is different from f_3. In f_7, a flit can be counted several times if the physical channel it goes through are shared by a number of paths.
- $f_5 \mid f_3$: f_5 *constrained* by f_3, i.e., a mapping is considered better only if it has smaller values of f_5 *and* f_3.
- $f_7 \mid f_3$: f_7 constrained by f_3.

It is obvious that finding true optimal mappings with respect to each of the above cost functions is NP-hard, i.e., there are no known polynomial time algorithms. Also, finding optimal mappings with respect to them is not very meaningful since the cost functions themselves are not the actual performance measure. Therefore, our goal is to obtain good sub-optimal mappings with respect to each cost function with a reasonable computing time, and the mappings will perform well at run-time and show significant improvements over random mappings. We will adopt the simulating annealing method for this purpose.

The termination of a simulated annealing process is generally decided by the following parameters: the initial temperature, the freezing point, the temperature updating function, and the exit criteria at each temperature. For each tested cost function, we carefully select these parameters so that for a given input traffic pattern and traffic density, the optimization process will terminate in approximately the same number of *trials*, denoted by n_T. A trial is defined as an instance of randomly choosing two modules and exchanging their positions, followed by the evaluation of the cost function. On a Sun IPX workstation, the compiled C program requires approximately 20 seconds of CPU time for 500 trials. For each given n_T, only those inputs that the optimization process terminates after $n_T \pm 10\%$ trials are used. The resulting mappings are collected and their average performance is calculated. Note that we do not artificially force the simulated annealing process to stop. Instead, we choose the parameters carefully and discard inputs which can lead to early or late terminations when using any of the above cost functions. By doing this, we can ensure fairness in comparing the effectiveness of cost functions.

3 Performance Evaluation

The mappings optimized with respect to the various cost functions are fed into a network simulation program. Under the following assumptions, we developed the program that simulates the flit-level communication behavior. The simulation results presented here were obtained using the following parameters:

- Transferring a flit between two nodes via a physical channel takes one unit of time.
- At any instant of time, all flits that have been allocated channels are transferred synchronously in a single physical channel cycle.
- Each virtual channel is assigned a single-flit buffer.
- The default packet-scheduling policy is FIFO, and the default flit-multiplexing method is RR with demand-driven (DD) allocation.
- Both w- and f- meshes are of size 16×16. Since performance trends are similar for f-meshes and w-meshes for the same P, unless stated otherwise, only the data obtained with w-meshes are plotted. The number of communicating modules is fixed at 256, i.e., the same as the number of nodes in the network. The default number of virtual channels is $v = 4$.
- Unless stated otherwise, all packets are 20 flits long. During the task execution, The probability, *density*, that node i sends a packet to node j in the uniform traffic pattern is 0.01. In a 16×16 network, the total number of concurrent packets during a mission is $\approx 0.01 \cdot (16^2 - 1)^2$.
- Unless stated otherwise, the traffic pattern is uniform. In hot-spot traffic, 5 hot spots in the network are randomly chosen, with *density* $= 0.5$ between any node and each of the hot spots. The default value of ΔT is 0.
- Each data point is obtained by averaging results from 10,000 mappings. Deviation from the mean values is found to be small ($< 5\%$).

In Figs. 1 and 2, the makespans of average latency of mappings optimized with respect to the various cost functions after ≈ 500 trials, (i.e., $n_T = 500$), are compared for different values of v. The performance of f_1 and f_2 are found to be very close to that of f_4 and f_6, and hence are not shown. f_1 and f_2 are only found to be effective in the case of f-meshes with large v's and small *density* values. This can be attributed to the fact that only in these situations makespan and latency estimates are more accurate.

From the results shown, it is obvious that f_7, $f_5 \mid f_3$ and $f_7 \mid f_3$ perform better than the other cost functions in this case. Mappings optimized with respect to these functions are also more resilient to the change of v's. On the other hand, mappings optimized with some functions (e.g., f_4 and f_6) perform well with small v's but become worse with larger v's. In the case of f_4, mappings optimized with respect to it improve over the random mappings when $v \leq 6$, but actually perform worse than random mappings when v gets larger. A similar behavior can also be observed from mappings optimized with the other mini-max type cost function, f_6, though to a less pronounced degree.

In Figs. 3 and 4, the performance of mappings optimized with the various cost functions under uniformly-distributed traffic are evaluated with variable n_T's. The number of virtual channels is fixed at $v = 4$. A good cost function should demonstrate a more predictable behavior, i.e., better performance measurements and less fluctuations when n_T is increased. Note that for each plotted curve, $n_T = 0$ corresponds to random mappings. Among the cost functions investigated, f_1, f_2, f_4 and f_6 all demonstrate highly unpredictable behaviors with increasing n_T. With more computing effort, mappings optimized with these cost functions can often *worsen* performance. This phenomenon is especially prominent with the makespan measurement. On the other hand, f_3, f_7, and their related functions like $f_5 \mid f_3$ and $f_7 \mid f_3$, all have more predictable performance, at least up to a much larger n_T than other cost functions. In makespan measurements, f_3 shows continual improvement of mappings up to $n_t = 3000$, and $f_5 \mid f_3$ can improve up to $n_T = 4000$. While $f_7 \mid f_3$ is found to improve mappings continually up to $n_T = 10,000$. For average latency measurement, there are less fluctuations for all cost functions. However, mappings optimized with most cost functions stop making noticeable improvement after $n_T \geq 1000$. Only f_7, $f_5 \mid f_3$ and $f_7 \mid f_3$ show continual improvement for $n_T > 1000$, while $f_7 \mid f_3$ shows improvement even when $n_T > 8000$.

Figs. 5 and 6 compare the performance of mappings optimized with various cost functions under hot-spot traffic. For makespan measurement, almost each cost function becomes less predictable under hot-spot traffic. Except $f_5 \mid f_3$ and $f_7 \mid f_3$, mappings optimized with all other functions cease to improve after $n_T > 500$. $f_5 \mid f_3$ starts to show fluctuations after $n_T > 4000$. On the other hand, $f_7 \mid f_3$ still shows predictable improvements after $n_T > 5000$.

From the above results, we can conclude that mappings optimized with respect to $f_7 \mid f_3$ have the most predictable improvement under various traffic patterns. Thus, we will focus on evaluating this particular function.

Given the same P, the effect of increasing ΔT is shown in Figs. 7 and 8. Mappings are optimized with $f_7 \mid f_3$ after 5,000 trials. It is obvious that mappings optimized with $f_7 \mid f_3$ still improve over random mappings with significant margins for large ΔT's. It is found that even with $\Delta T = 250$, the margin of improvement is still more than 20% for both makespans and average latency measurements. Note that, for random mappings, the makespan decreases monotonically with increasing ΔT up to 180, showing that even when packets in P arrive in such a large time window, the network is still saturated. On the other hand, for mappings optimized with $f_7 \mid f_3$, the network becomes less congested when $\Delta T > 120$ and makespan tilts upward slightly with increasing ΔT's.

In [3], we have shown that by employing appropriate packet-scheduling policies and flit-multiplexing methods, the performance of a virtual-channel network under concurrent communication traffic can be greatly improved. Here we will demonstrate that by applying these run-time flow-control mechanisms, the performance of mappings which are already optimized with $f_7 \mid f_3$, can be improved further. In Figs. 9 and 10, the performance measurements are shown for mappings optimized with $f_7 \mid f_3$ when executed on systems with various packet-scheduling and flit-multiplexing combinations. "SRBF" denotes the packet-scheduling policy which gives a higher priority to the packet with the smallest remaining bandwidth. "SRBP" denotes the flit-multiplexing method giving a higher priority to the same type of packets as in SRBF. These two schemes are shown in [3] to perform particularly well. Also, to prevent deadlock, CTS lookahead is implemented with SRBP multiplexing.

These flow-control mechanisms can still improve the performance of mappings significantly. Though using SRBF scheduling alone can introduce some performance fluctuations when n_T is increased, it can still improve makespans by at least 12% and average latency by at least 10%. The combination of SRBF and SRBP can further improve the performance, especially the average latency. Also note that, when these flow-control mechanisms are used, the margin of improvement with increasing n_T's is narrowed. For example, mappings found with $n_T = 5000$ still outperform $n_T = 1000$, but when compared with the case using only FIFO-RR, the margin is greatly reduced. This shows that by using proper run-time flow controls, we may save some computing effort on finding optimized mappings.

In Figs. 11 and 12 we show the effect of applying the mapping optimization process and flow-control mechanisms on the performance of one set of communicating modules under uniform and hot-spot traffic, respectively. The mapping is optimized with $f_7 \mid f_3$ and $n_T = 5000$. It can be observed that given a mapping, different flow-control mechanisms will result in different rates of "energy" (remaining bandwidth) dissipation. Better flow-control not only results in a higher rate, but also a more linear behavior, and hence, a more predictable task communication response time. Furthermore, in the presence of hot-spot traffic, a good flit-multiplexing method like SRBP can reduce makespan dramatically by reducing the time the system spends in non-saturated regions, as shown in Fig. 12.

On the other hand, the mapping optimization process leads to lower "initial energy,", and reduces the time needed to dissipate it. Note that it can work independently of the flow-control mechanisms, and their improvements on the performance can be additive. It is also interesting to note that the amount of initial bandwidth is a good indication of the quality of mappings, especially when ΔT is small. In most of our inputs used here, when $\Delta T < 70$, a mapping with a smaller initial bandwidth almost always has a better makespan and average latency measurements. However, in most cases, given the same computing time, mapping optimized with f_3 actually has a higher initial bandwidth than $f_7|f_3$. The reason for this is that using f_3 alone, the simulated annealing process can be "trapped" in a local optimum much more quickly than using $f_7 \mid f_3$.

4 Conclusion

In this paper, we have addressed the problem of mapping concurrently-communicating modules into mesh-connected multicomputers equipped with wormhole switching and virtual channels. Our objective is to optimize the makespan and average latency of these packets exchanged among modules. It has been shown that direct optimization of the performance objective is not practical. We investigated several simplified cost functions for the simulated annealing method. The effectiveness of these proposed cost functions are compared by using a flit-level simulation program to access the actual run-time performance of the mappings optimized with each cost function when approximately the same amount of computing time is given. The cost function $f_7 \mid f_3$, has been found to be quite effective. Mappings optimized with it have been shown to be consistently outperform the others. Also, performance of mappings can be continually improved with the increase of computing time. We also showed that the run-time performance of optimized mappings can be further improved when on-line flow-control mechanisms are used.

References

[1] M. G. Norman, "Models of machines and computation for mapping in multicomputers," *ACM Computing Surveys*, vol. 25, no. 3, pp. 263–302, September 1993.

[2] W. J. Dally, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[3] B.-R. Tsai and K. G. Shin, "Sequencing of concurrent communication traffic in mesh multicomputers with virtual channels," in *Proc. of the 23-rd International Conference on Parallel Processing*, pp. 126–133, August 1994.

[4] B.-R. Tsai and K. G. Shin, "Communication–oriented assignment of task modules in hypercube multicomputers," in *Proc. 12–th Int'l Conf. on Distributed Comput. Syst.*, pp. 38–45, June 1992.

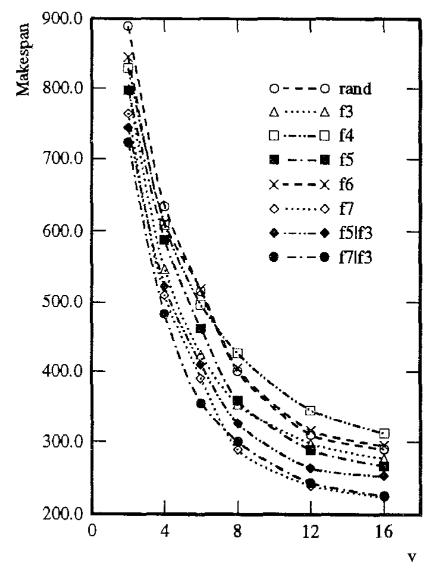

Figure 1: Makespan comparison of mappings optimized with various cost functions.

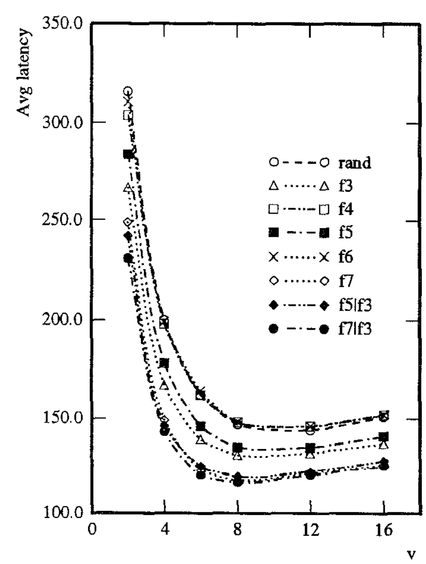

Figure 2: Average latency comparison of mappings optimized with various cost functions.

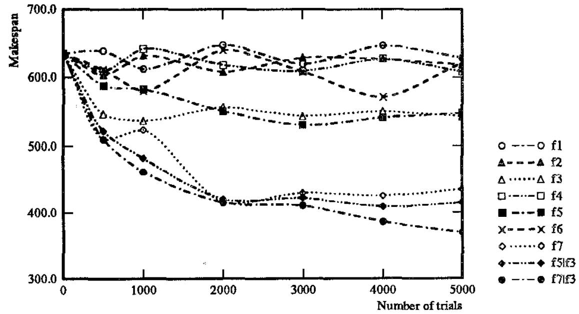

Figure 3: Makespan comparison of mappings optimized with various cost functions, uniform traffic.

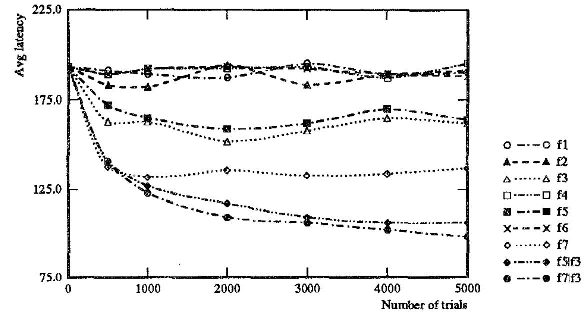

Figure 6: Average latency comparison of mappings optimized with various cost functions, hot-spot traffic.

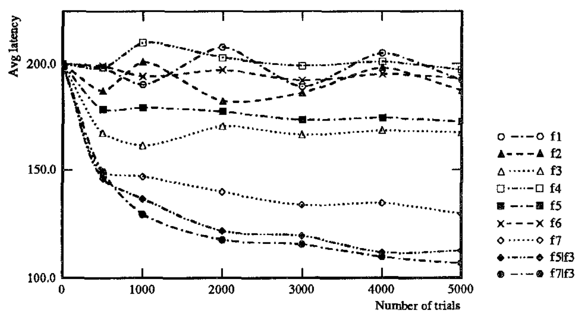

Figure 4: Average latency comparison of mappings optimized with various cost functions, uniform traffic.

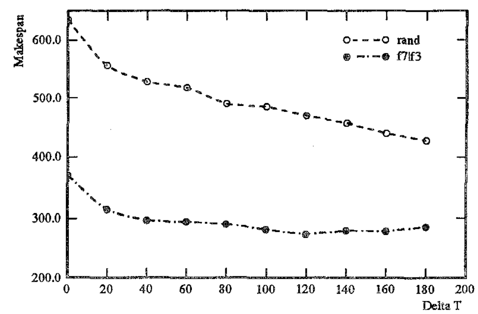

Figure 7: Makespan of mappings optimized with $f_7 \mid f_3$ versus varying ΔT.

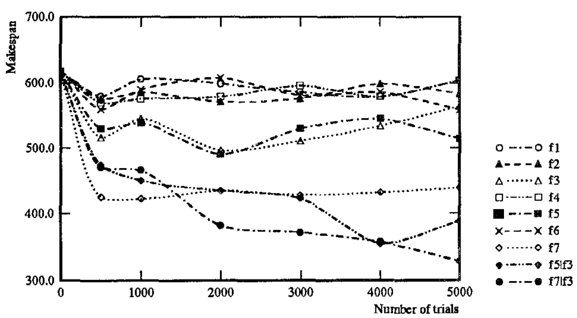

Figure 5: Makespan comparison of mappings optimized with various cost functions, hot-spot traffic.

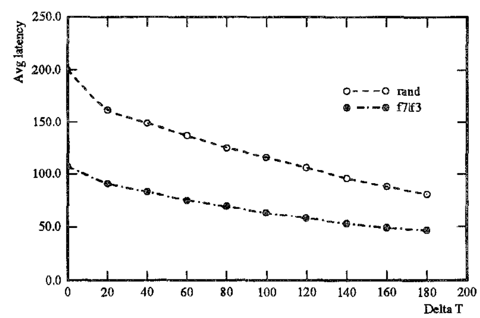

Figure 8: Average latency of mappings optimized with $f_7 \mid f_3$ versus varying ΔT.

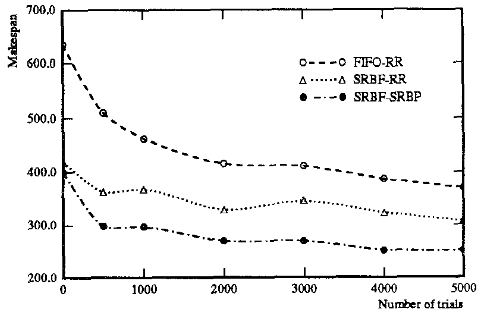

Figure 9: Makespan of mappings optimized with $f_7 \mid f_3$ under different flow-control strategies.

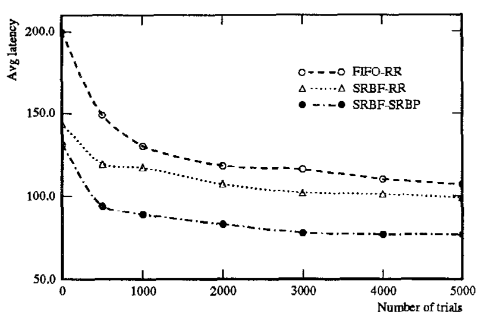

Figure 10: Average latency of mappings optimized with $f_7 \mid f_3$ under different flow-control strategies.

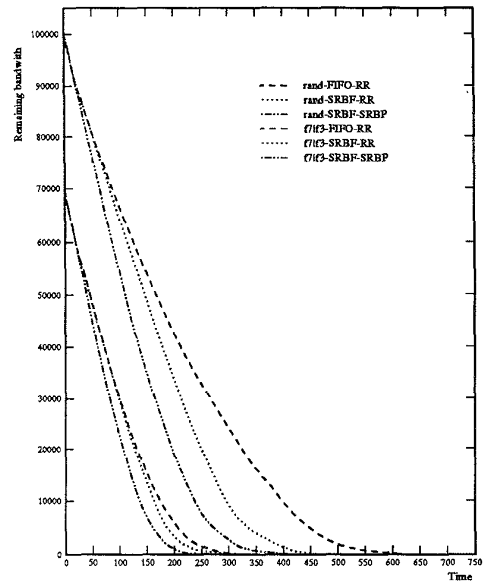

Figure 11: Plot of remaining bandwidth versus time, uniform traffic.

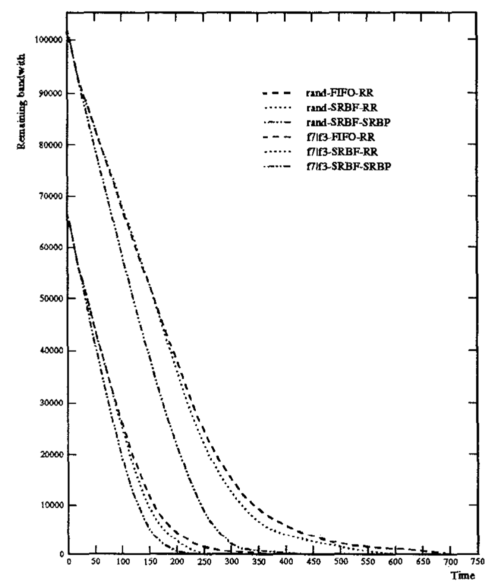

Figure 12: Plot of remaining bandwidth versus time, hotspot traffic.