Growing Squares: Animated Visualization of Causal Relations

Niklas Elmqvist[*] Philippas Tsigas[†]

Department of Computing Science
Chalmers University of Technology
SE-412 96 Göteborg, Sweden

Abstract

We present a novel information visualization technique for the graphical representation of causal relations, that is based on the metaphor of color pools spreading over time on a piece of paper. Messages between processes in the system affect the colors of their respective pool, making it possible to quickly see the influences each process has received. This technique, called Growing Squares, has been evaluated in a comparative user study and shown to be significantly faster and more efficient for sparse data sets than the traditional Hasse diagram visualization. Growing Squares were also more efficient for large data sets, but not significantly so. Test subjects clearly favored Growing Squares over old methods, naming the new technique easier, more efficient, and much more enjoyable to use.

CR Categories: D.1.3 [Programming Techniques]: Concurrent Programming; D.2.5 [Software Engineering]: Testing and Debugging; H.5.1 [Information Systems]: Multimedia Information Systems—Animations; H.5.2 [Information Systems]: User Interfaces; I.3 [Computer Methodologies]: Computer Graphics

Keywords: causal relations, information visualization, interactive animation

1 Introduction

The notion of cause and effect is pervasive in human thinking and plays a significant role in our perception of time. Software systems, in particular parallel and distributed ones, are permeated by this *causality*, and the human mind is especially well-suited to detect instances of this concept. For instance, we can all easily trace a rolling billiard ball back to the ball that struck it and set it in motion. However, as the number of actions and reactions in a system grows, it quickly becomes difficult to follow and gain an understanding of its general flow. Accordingly, a billiard table where all 16 balls are moving is impossible to comprehend fully in real-time. Traditional visualizations, notably directed acyclic graphs (DAGs) and Hasse diagrams (also called time-space diagrams), can allay this problem somewhat, but become inefficient as a system grows due to their fine granularity; users can see the individual relations, but not get a good picture of the system as a whole.

This paper presents a novel visualization technique based on animation, colors and patterns to provide an alternate graphical representation of causality in a system that facilitates quick overview. By representing each process (i.e. active entity) in the system as a color-coded square, laid out in a suitable way, we intuitively "grow" each process as time progresses and have the events that causally relate them affect their coloring, somewhat akin to how color pools would spread out on a piece of paper (see Figure 1). We have developed a visualization framework for causal relations that allows us to compare the Growing Squares method with traditional Hasse diagrams (see Figure 2). In addition, this framework allows the user to select different visualizations of causal relations on the fly.

A formative evaluation, using a focus group consisting of researchers working on distributed systems, was conducted at the onset of the project in order to identify the tasks associated with causal relations and shape the design of the visualization. Furthermore, a user study was performed at the end to ensure the validity of our findings. Our results show that the Growing Squares method is significantly faster and more efficient than Hasse diagrams for sparse data sets, and more efficient also for dense data sets, but not significantly so. In addition, test subjects clearly favored Growing Squares over Hasse diagrams for all analysis tasks performed. Overall, the subjective ratings of the test subjects showed that the Growing Squares method is easier, feels more efficient, and is more enjoyable to use than Hasse diagrams.

In modern usage, the notion of causality is associated with the idea of something (the cause) producing or bringing about something else (its effect). In general, the term "cause" has a broader sense, equivalent to an explanatory or reasoning tool. Identifying causal relations in a complex system can be the first step towards understanding the underlying mechanisms that determines the system's laws. As such, causal relations cover a wide variety of software domains where causality might be of importance.

More specifically, causal relations play a central role in parallel and distributed software development; they are used extensively in collecting and analyzing data needed to evaluate the correctness and the performance of such programs. Visualization and debugging environments for parallel and distributed software commonly offer animated views of causal relations. The extensive use of causal relations in these fields comes from the fact that in such software systems, only partial views of the complete set of programs can be taken at any time. Causally consistent views are the only way to construct a complete view of a whole distributed or parallel software system. For example, causal relations are

[*]elm@cs.chalmers.se
[†]tsigas@cs.chalmers.se

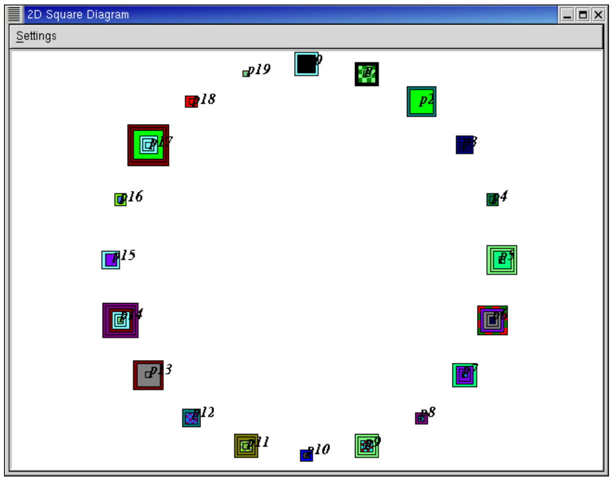


Figure 1: Growing Squares visualization with 20 processes.

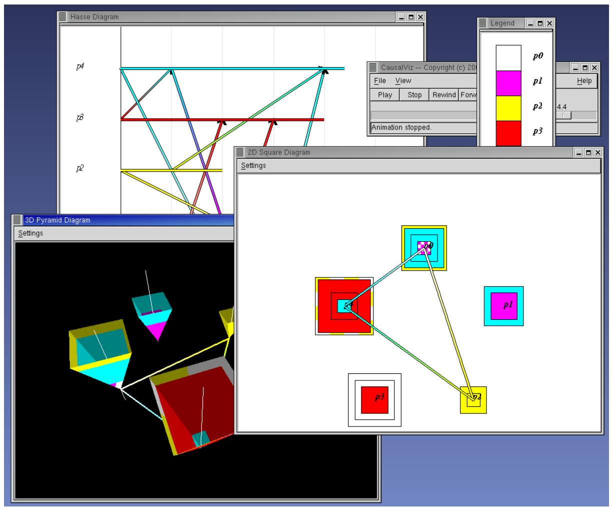


Figure 2: The CausalViz application.

used (i) in distributed database management to determine consistent recovery points; (ii) in distributed software systems for determining deadlocks; (iii) in distributed and parallel debugging for detecting global predicates and detecting synchronization errors; (iv) in monitoring and animation of distributed and parallel programs to determine the sequence in which events must be processed so that cause and effect appear in the correct order; and (v) in parallel and distributed software performance to determine the critical path abstraction: the longest sequential thread, or chain of dependences, in the execution of a parallel or distributed program. Effective visualization of causal relations will be of benefit to all these activities.

The next section presents a theoretical treatment of causal relations in general, immediately followed by related work. Then, we describe the Growing Squares visualization technique and the user study we conducted to evaluate it. Results, discussion and conclusions follow at the end.

2 Causal Relations

A causal relation is the relation that connects or relates two items, called *events*, one of which is a cause of the other. Obviously, for an event to cause another, it is not sufficient that the second merely happens after the first. However, it is well accepted to state that this is necessary, and temporal order can be relied on to explain the asymmetrical direction of causal relations[1]. All events connected in the causal relation are part of a set of *processes*, labelled $P_1, \ldots, P_N$, each of which can be thought of as a disjoint subset of the set of all events in a system. Events performed by the same process are assumed to be sequential; if not, we can split the process into sub-processes. Thus, it is convenient to index the events of a process P_i in the order in which they occur: $E_i = e_1^i, e_2^i, e_3^i, \ldots$

For our purposes, it suffices to distinguish between two types of events; *external* and *internal* events. Internal events affect only the local process state. An internal event on processes P_i will causally relate to the next event on the same process. External events, on the other hand, interconnect events on different processes. Each external event can be treated as a tuple of two events: a *send* event, and a corresponding *receive* event. A send event reflects the fact that an event, that will influence some other event in the future, took place and its influence is "in transit"; a receive event denotes the receipt of an influence-message together with the local state change according to the contents of that message. A send event and a receive event are said to correspond if the same message m that was sent in the send event is received in the receive event.

We now formally define the binary causal relation $\rightarrow$ over all the events of the system E ($\rightarrow \ \subseteq E \times E$) as the smallest transitive closure that satisfies the following properties [Lamport 1978]:

1. If $e_i^k, e_i^l \in E_i$ and $k < l$, then $e_i^k \rightarrow e_i^l$.
2. If $e_i = send(m)$ and $e_j = receive(m)$, then $e_i \rightarrow e_j$ where m is a message.

When $e \rightarrow e'$, we say e causally precedes e' or e caused e'. Causal relations are irreflexive, asymmetric, and transitive.

3 Related Work

The traditional way to visualize causal relations are Hasse or time-space diagrams. Figure 9 shows an example of a time-space diagram for a system comprised of five processes, where the progress of each process is described by a directed horizontal line, the process line. Time is assumed to move from left to right. Events are symbolized by dots on the process lines, according to their relative order of occurrence. Messages are shown as arrows connecting send events with their corresponding receive events. Visualizations of causal relations in the form of such time-space diagrams are currently quite standard in visualization and debugging platforms for parallel and distributed systems, and the number of such platforms is too large to allow mentioning them all. One of the first of the new generation of visualization tools to include the time-space diagram was the Voyeur [Socha

[1]It has been argued that not even this is necessary, and that both simultaneous causation and "backwards causation" (effects preceding their causes) are at least conceptually possible. This, on the other hand, causes problems when considering the asymmetric nature of causal relations.

et al. 1989] system, which provided a framework for defining various animation views for parallel algorithms. The TOPSYS [Bemmerl and Braum 1993] environment includes various standard concurrency visualizations integrated with the debugging and performance analysis tools of the system, with time-space visualization being one of them. Going one step further, the conceptual visualization model of VADE [Moses et al. 1998] system is based on the causal relation notion. Also of interest is LYDIAN [Koldehofe et al. 1999], an educational visualization system, which by default constructs the time-space diagram for every algorithm implemented in the system. Kraemer and Stasko [1998] describe the essential characteristics of toolkits for visualization of concurrent executions, and introduce their own system, called Parade. For the purpose of our study, the Hasse visualization used in Figure 9 is very similar to the time-space visualization view from the ParaGraph system [Heath 1990; Heath and Etheridge 1991] and its adaption in the PVaniM tool [Topol et al. 1998], as well as the Feynman or Lamport views from the Polka animation library [Stasko and Kraemer 1993].

Ware *et al.* [1999] presented a new visualization construct called a *visual causality vector* (VCV) that represents the perceptual impression of a causal relation and employed animation to emphasize this relation in a directed acyclic graph. Three different VCVs were introduced based on different metaphors: the pin-ball metaphor, where the VCV is a ball that moves from the source to the destination node, striking the destination and making it oscillate; the prod metaphor, where the VCV is a rod that extends from the source to prod the destination; and finally a wave metaphor, where the VCV accordingly is an animated wave that moves towards the destination node. However, while these constructs are certainly an improvement over a simple DAG representation of causal relations, they do nothing to battle the complexity of large systems with many nodes and relations. In fact, Ware's primary contribution is the investigation of timing concerns for the perception of causality for users, not the visualization technique *per se*. It might still be interesting to incorporate Ware's VCVs into our system in some form.

4 Growing Squares

As described above, there is surprisingly little work on visualizations of causal relations besides various implementations of Hasse diagrams, a fact which is especially curious in light of the shortcomings of Hasse diagrams for understanding a distributed system. The fine granularity of Hasse diagrams defeat their use as overview tools, and they transfer the burden of maintaining transitive relations to the user herself. This means that a user studying the information flow in a distributed systems visualized using a Hasse diagram might potentially have to backtrace every single message and process in order to get a clear picture of the influences in the system.

The Growing Squares visualization technique was designed to help the user quickly get an overview of the causal relations in a system by making use of animation, color and patterns in an intuitive way. The visual metaphor of the technique is that of "pools" of color spreading on a piece of paper as time progresses, each color and pool representing a specific process or node in the system. Messages in the system are shown as "channels" from one pool to another. Each color pool will start growing at the time its corresponding process is started, and accordingly stop growing when the process stops executing events. The channels representing messages from one process to another naturally carries the color of its source with it, resulting in the destination pool receiving this color as well. However, like age rings on a tree, the color of the new influencing process will only be present in the destination process starting from when the message is received.

Figure 3 gives an example of a system with two processes, P_0 and P_1, colored blue and white, respectively. The color pools are represented as 2D squares which grow over time. At a certain time t, P_0 sends a message to P_1 (denoted by the arrow in the figure), establishing a causal relation between P_1 and P_0. For all times $t' > t$, the color pool of process P_1 now shows this influence from the blue P_0 by means of a checkered pattern combining the two colors.

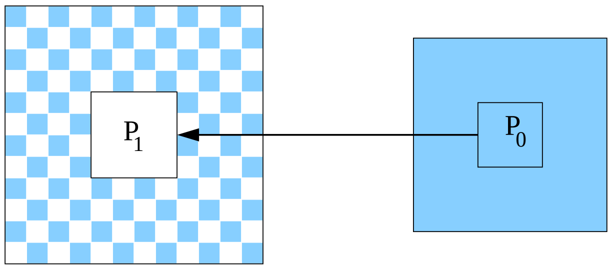


Figure 3: Simple example of the Growing Squares technique with two processes.

In order to visualize the transitive property of the causal relation (see the previous section), a similar color pattern scheme is used. In Figure 4, process P_1 is sending a message to P_2 (colored red) *after* having been influenced by a message from P_0. Now, both the color of the source process (white from P_1 itself) and any of its existing influences at the time of sending the message (blue from P_0) are transferred to P_2, making its texture from this time and onwards be a checkered pattern of all of the three colors. It is now easy to see that P_2 is causally related to both P_0 and P_1.

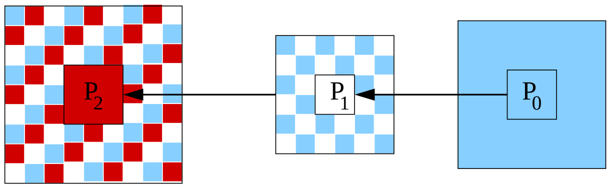


Figure 4: Transitivity property of causal relations using Growing Squares.

Multiple influences from the same source process will increase the amount of the source process's color in the texture of the destination process. Even if the checkered pattern makes it difficult to see the exact ratio, this fact can nevertheless be used as a visual indication that multiple influences have occurred.

Having foregone a traditional timeline, the Growing Squares method is dependent on animation to allow the user to view the entire execution of the system under study. Starting at $t = 0$, the user can advance the time in the system to observe the system execution in chronological order, or choose to view the situation at specific points in time. This is another radical difference from Hasse diagrams; Hasse diagrams are static in nature and do not benefit much from animation, whereas Growing Squares are dynamic and rely on animation to present the full data set to the user.

Figure 10 (also the color plate) shows an example sequence consisting of 5 processes in a distributed system visualized using the Growing Squares technique (this is the same system shown using a Hasse diagram in Figure 9). The state of the visualization is here shown for each discrete time unit (in practice, the animation is fluid and continuous between the time steps) starting at $t = 1$ and ending at $t = 5$, the end of the execution. Processes are laid out in a clock-wise fashion with P_0 at the top. Screenshot (a) at $t = 1$ shows how P_1 sends a message to P_0, starting it (it has zero size up until this time), and (b) at $t = 2$ depicts the two colors (green and black) in the process square of P_0. In the same screenshot, P_4 sends a message to P_1, causing P_1 in (c) at $t = 3$ to hold influences from both P_4 as well as indirectly from P_3 (i.e. an example of the visualization of transitivity in the Growing Squares visualization). In (d) at $t = 4$, two messages originating from the otherwise isolated P_2 reach P_0 and P_4, its blue color showing in the outer square of these two processes in snapshot (e) at $t = 5$.

4.1 Design Decisions

In order for the Growing Squares visualization to be effective, users must be able to easily distinguish between the individual process colors in the system under study. Selecting a suitable color scale is thus an important aspect of the method, and we investigated the use of perceptually uniform color scales such as LOCS [Levkowitz and Herman 1992; Levkowitz et al. 1992] for this purpose. However, we found that the continuous nature of LOCS was not well-suited to our problem since it made distinguishing between adjacent colors difficult, and the scale itself included an inordinate amount of dark colors. Instead, we opted for a simple color scale with the individual colors uniformly distributed over the RGB spectrum.

One of the central features of the presented visualization technique is that it draws process squares with the checkered patterns containing all the colors of the processes that have influenced the process. If the number of influences is large, the on-screen space allocated for each color will be very small and thus hard to distinguish (see [Wyszecki and Stiles 1991] for in-depth information on color perception). In order to still allow the visualization to be effective, we need a zoom function that allows the user to effortlessly view the graphical representation at different magnification levels. We have implemented a simple continuous zoom mechanism for this purpose; in the future, it may be extended to borrow techniques from the Pad [Perlin and Fox 1993] zoomable user interface and its descendants [Bederson et al. 1996; Bederson et al. 2000].

It might be argued that using circles instead of squares would have been more in keeping with the metaphor of color pools spreading on a piece of paper. Our original intention was also to use circles, but we ultimately chose squares for a number of reasons: (i) the larger area of squares facilitates color recognition better than circles, (ii) the layout of process squares into grids is easier (no wasted space), and (iii) squares are faster to render and easier to texture map (besides, we felt it was more logical to have checkered squares rather than checkered circles).

The Growing Squares visualization makes use of animation to display the dynamic execution of the system under study. While it certainly is possible to maintain all message arrows and just draw the visualization at full time, this would result in many of these messages coinciding (as in Hasse diagrams) and thus being hard to separate from other messages, as well as being impossible to associate with a specific time. Animation solves these issues in a natural way.

Another design aspect of the Growing Squares visualization is finding suitable layout methods for arranging the individual processes. Many such layout strategies exist. For instance, if the data set represents the execution of a distributed algorithm in a network, the geographical location of the individual nodes can be used to position the squares in the visualization. Other alternatives include simple grid and circular layouts (see Figure 1) which may serve to minimize the amount of coinciding message arrows to greater or lesser extent. Beyond these simple layout schemes, there exists a wide array of advanced graph layout algorithms that could be incorporated into our scheme [Battista et al. 1999]. In this paper, we chose to ignore this aspect and selected a simple circular layout scheme that has the advantage of avoiding message arrows coinciding with each other or passing over processes; in the future, we might look into making use of a graph layout library such as [Alberts et al. 1997] in the system.

4.2 Analysis Tasks

At the onset of our investigation into visualization of causal relations, we organized a formative evaluation of these concepts using a focus group consisting of researchers from our university working on distributed systems. The purpose of this evaluation was to determine the analysis tasks a causal relation visualization should support. The discussion in the focus group unearthed many perceptions on causal relations from professionals in the field, and were instrumental in tailoring the design of the visualization technique to the types of analyses typically performed using the causality concept.

Our main finding was that users in general want to use causality visualizations for gaining an overview of a distributed system. Here, we loosely define *overview* as a self-contained graphical representation providing the information context necessary for quickly understanding the basic trends of the data. We refer to these types of analysis tasks as *overview tasks*. Furthermore, users indicated that understanding the information flow and influence relationships in a system is the most important activity performed when analyzing a distributed system, and the Growing Squares visualization was accordingly designed specifically with this purpose in mind.

Here follows a short description of the most common overview tasks, derived from our formative evaluation, performed when analyzing distributed systems, along with an account of how our technique aids the user in performing them.

4.2.1 Duration Analysis

The duration (or lifetime) of a process tells how much CPU time the process has consumed, and processes with a long life also tend to have a large influence in the system. Growing Squares affords fast and easy recognition of the set of processes that have the longest lifetimes simply by comparing relative sizes at the end of the execution. Accurately determining the single process with the longest duration is more difficult, however.

4.2.2 Influence Analysis

The number of messages a process sends to other processes can be used as a measure of the relative importance of the

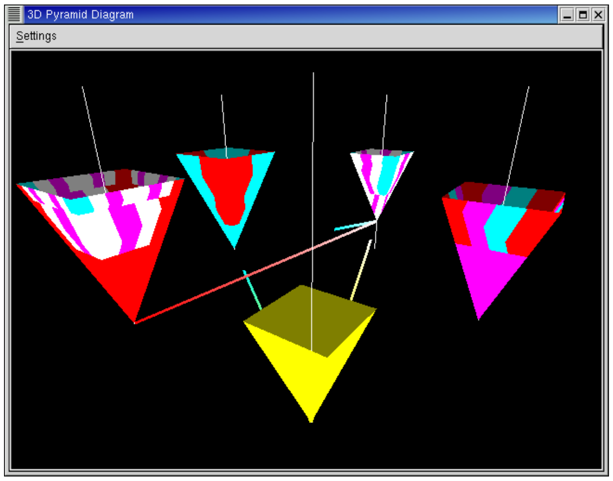


Figure 5: 3D Growing Pyramids visualization with 5 processes.

process, since each message will exert some (unknown) influence on the recipient. Our technique is especially well-suited to helping the user quickly get a feeling for the most influential processes as well as the processes that have been influenced the most. Again, accurate determination of the single most influential process is difficult (especially for large systems where process colors tend to look similar).

4.2.3 Inter-Process Causal Relations

In some cases, it is necessary to check whether there is a causal relation between two processes P_i and P_j, i.e. whether there exists events $e^i \in E_i$ and $e^j \in E_j$ so that $e^i \to e^j$. However, due to the transitive nature of the causal relation, this may often be difficult using Hasse diagrams since it may be necessary to recursively trace back all messages received by P_j to see whether any of them originate from P_x. Our method directly shows the influences a process has, including transitivity information, so it is only a matter of seeing whether the color of P_i exists in P_j's square.

4.3 Extension to 3D

One of the drawbacks of the Growing Squares technique is that it does not offer accurate information on the starting times for the respective processes. The size of the square can be used to compare the relative life times of different processes, but one needs to use the animation controls to deduce which of them started first. To address this problem, we have designed a straightforward extension of Growing Squares to 3D where the time parameter is mapped to the vertical axis (see Figure 5). Thus, processes now grow on the Y axis as well, creating pyramids instead of squares (the technique is accordingly called Growing Pyramids). However, for the purposes of this paper, we focus strictly on the 2D version of the technique.

4.4 Implementation

In order to test the Growing Squares technique and to subsequently be able to perform a user study on its effectiveness, we have implemented a general application for the visualization of causal relations called CausalViz (see Figure 2). The application is implemented in C++ on the Linux platform and uses the Gtk+/Gtk– widget toolkits for user interface components as well as OpenGL for graphical rendering.

4.4.1 System Architecture

The architecture of the CausalViz application (see Figure 6) is based around a single partially ordered set (*poset*) representing the execution data under study. A number of visualization components observe this set and present graphical representations of the data (potentially allowing for the set to change during run-time). There currently exists three different visualizations, i.e. traditional Hasse diagrams, the 2D Growing Squares, and the prototype 3D Growing Pyramids.

Central in the system architecture is the *application manager* that creates all the other components, manages the graphical user interface (GUI), and performs loading of data files into the application (stored in a general XML format for partially ordered sets). In order to allow for the animation of events in the visualizations, there also exists a general *animation manager* thread that the visualization components can use to smoothly interpolate values in the poset with respect to time.

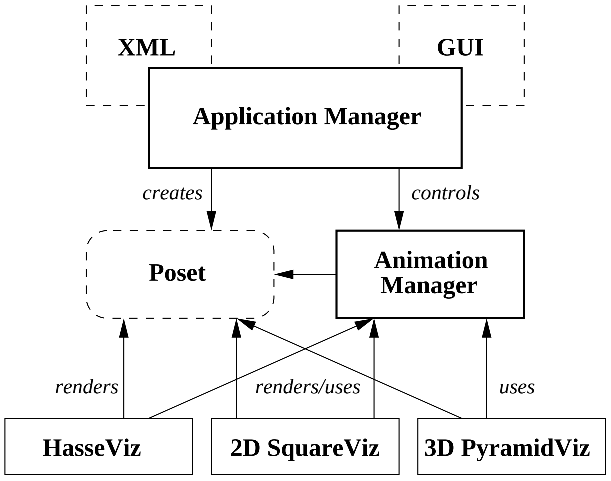


Figure 6: CausalViz system architecture.

4.4.2 Hasse Visualization

In order to provide a comparative measure to our new method, we implemented the traditional Hasse visualization technique in the CausalViz application. Many such implementations exist, for instance the time-space diagram from the ParaGraph system [Heath 1990; Heath and Etheridge 1991] and its adaptation in the PVaniM tool [Topol et al. 1998], as well as the the Feynman or Lamport views from the Polka algorithm animation library [Stasko and Kraemer 1993]. Figure 9 is a screenshot of our implementation.

4.4.3 Poset Management

System execution traces are stored in a general XML file format for partially ordered sets. Here, a process P_i is represented by the subset $E_i \subseteq E$ of all the events in the system belonging to the process and a set of messages M. Messages are partial orderings between events in different subsets (processes), and can thus be represented by pairs of events, i.e. $M \subseteq E \times E$. It is then up to the application to compute the minimal transitive closure for the poset.

In the CausalViz application, the transitive closure is computed using a modified topological sort [Cormen et al. 2001]. The objective of the algorithm is two-fold: (i) to derive the transitivity information for each event (i.e. the processes which have influenced it so far) and (ii) to assign the event to a discrete time slot. This is done by greedily consuming sequential events in each subset (i.e. process) of the poset until reaching an event with unresolved dependencies (i.e. a partial ordering to a previously unvisited event). When this happens, the algorithm moves on to the next process to continue from where it last left off. This is repeated until all events in the system have been visited. The current influence of each event is easily maintained and updated during this process, and illegal cyclic dependencies are trivially detectable by checking whether the algorithm has cycled through all process without visiting any new events.

5 User Study

Our hypothesis was that the Growing Squares technique is faster and more efficient at quickly providing an overview of the causal relations in a distributed system, and that the new technique scales better with system size than traditional methods. To test this, we conducted a formal comparative user study of the old Hasse diagram visualization and our new Growing Squares technique. The focus of this user study was to evaluate user performance of the "overview tasks", tasks associated with general comprehension of how a system works that were derived from our formative evaluation. We also wanted to get a subjective assessment of the two methods.

5.1 Subjects

Twelve users, four of which were female, participated in this study. All users were carefully screened to have good computer skills and basic knowledge of distributed systems and general causal relations. In particular, knowledge of Hasse diagrams was required. Subject ages ranged from 20 through 50 years old, and all had normal or corrected-to-normal vision.

5.2 Equipment

The study was run on a high-end Intel Pentium III 866 MHz laptop with 256 MB of memory and a 14-inch display. The machine was equipped with a NVidia Geforce 2 GO graphics accelerator and ran Redhat Linux 7.2.

5.3 Procedure

The experiment was a two-way repeated-measures analysis of variance (ANOVA) for independent variables "visualization type" with two levels (Hasse diagrams versus Growing Squares), and "data density", also with two levels. The two levels of data density were "sparse" and "dense" with 5 processes sending 15 messages and 30 processes sending 90 messages, respectively. The visualization type was a within-subjects factor, as was the data density. Each subject received the various task sets in different order to avoid systematic effects of practice.

The same set of four different data sets were used for all subjects. Two were geared at the sparse case with 5 processes and 15 messages (one for each visualization type), and two for the dense case with 30 processes and 90 messages (see Table 1). The traces were all generated using a heuristic algorithm to avoid users taking advantage of special knowledge about real system traces. In the case of deducing inter-node causal relations, care was taken to ensure that the complexity of this was equivalent for both task sets of each density.

Data Density	**Processes**	**Messages**
Sparse	5	15
Dense	30	90

Table 1: Experimental design. Both density and visualization factors were within subjects for all 12 subjects.

The evaluation procedure consisted of repeating overview tasks using Hasse diagrams and Growing Squares for first the sparse and then the dense data densities. The order of the visualization types was different for each subject to minimize the impact of a learning effect. The repeated tasks for each density and visualization type is summarized in Table 2. Prior to starting work on each task set, subjects were given the chance to adjust the window size and placement to their liking. Subjects were informed that they should solve the tasks quickly and focus on using the visualization to get an overview of the system trace. The completion of each task was separately timed, except for the tasks Causality 1-3, which were timed together.

Since we were targeting overview tasks, it was not necessary for subjects to find a precise answer to each exercise. Instead, it was deemed sufficient if subjects named one of the processes in the top 20 %[2] for each category; i.e. for 30 processes, it was enough to pick one of the six processes that were most the influential, long-lived or influenced ones for the answer to be counted as correct. Only the Causality 1-3 tasks required a totally accurate answer.

After having performed each task set for a density and visualization type, subjects were asked to give a subjective rating of the efficiency, ease-of-use, and enjoyability of the visualization technique. When all of the tasks were completed, the subjects responded to a final questionnaire comparing the two visualization techniques based on the previously-stated criteria.

Each evaluation session lasted approximately one hour. Subjects were given a training phase of ten minutes to familiarize themselves with the CausalViz application and the two visualization techniques. During this time, subjects were instructed in how to use the visualizations to solve various simple tasks.

6 Results

After having conducted the user study, we analyzed the resulting test data. The results can be divided into two parts; the objective performance measurement, and the subjective ratings of the test subjects.

6.1 Performance

The mean times of performing a full task set (i.e. four tasks) using the Hasse diagrams and the Growing Squares visualizations were 416.58 (s.d. 268.99) and 334.79 (s.d. 230.86)

[2]This number was somewhat arbitrarily chosen, partly because it was felt to be an acceptable margin of error, and partly because 20 % out of 5 processes for the sparse data set translates to finding the single correct process for each task.

Task	Comments	Measure
Duration	Find the process with the longest duration.	Time
Influence 1	Find the process that has had the most influence on the system.	Time
Influence 2	Find the process that has been influenced the most.	Time
Causality 1-3	Is process x causally related to process y?	Time
Q1	Rate the visualization w.r.t. ease-of-use (1=very hard, 5=very easy).	Likert
Q2	Rate the visualization w.r.t. efficiency (1=very inefficient, 5=very efficient).	Likert
Q3	Rate the visualization w.r.t. enjoyability (1=very boring, 5=very enjoyable).	Likert

Table 2: Repeated tasks for each density and visualization type.

seconds respectively. This, however, is not a significant difference ($F(1, 11) = 2.54$, $p = .139$). The main effect for density was strongly significant ($F(1, 11) = 30.99$, $p < .001$), with means for the sparse and dense conditions of 222.96 (s.d. 77.24) and 528.42 (s.d. 272.94) seconds. Figure 7 summarizes the mean task results for the two visualizations across the two densities; error bars show one standard deviation above and below the mean. The figure also shows that the mean time for the task set was higher for the Hasse method across all densities. For the sparse conditions the visualization type was significant ($F(1, 11) = 15.82$, $p = .002$), with mean values of 259.50 (s.d. 75.23) and 186.42 (s.d. 62.46) seconds for the Hasse and Growing Squares visualizations. The Growing Squares method also gave better results for dense conditions; the mean times in Hasse and Growing Squares were 573.67 (s.d. 302.96) versus 483.17 (s.d. 243.94) seconds. This, however was not a significant difference ($F(1, 11) = 1.03$, $p = .332$).

The subjects' comments revealed that one of the reasons for the absence of a statistically significant difference between visualizations in the dense condition was because of color similarities. Much time was spent by subjects matching colors to each other and looking up process numbers in the color legend.

Subjects made little use of the animation controls in the Growing Squares visualization except to play it through once at the beginning of each task to gain a picture of the data set. Only a few of the subjects actively moved the timeline back and forth to solve various subtasks, and most preferred to leave the time setting at the end of the execution.

The fixed (circular) layout algorithm used in the user study turned out to be limiting when it came to comparing the size (i.e. duration) of individual processes. Users remarked that it would have been useful to be able to click and drag processes to arbitrary positions to facilitate comparison as well as to group processes into semantic clusters (i.e. clusters of the same perceived type).

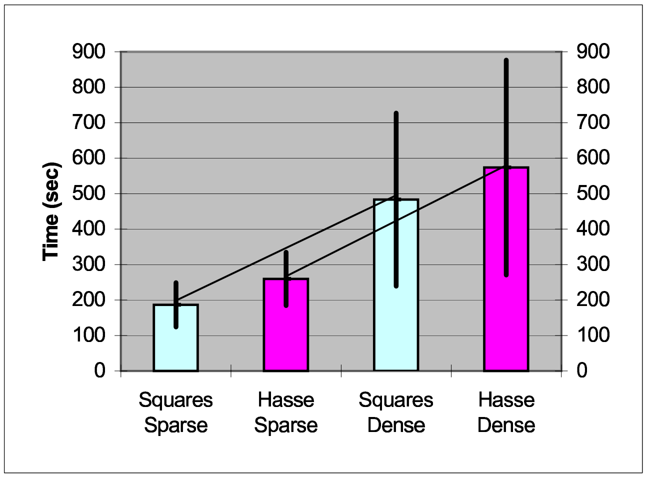


Figure 7: Mean task completion times for all tasks across the Hasse and Growing Squares methods and across levels of density. Error bars show standard deviations.

6.2 Subjective Ratings

The subjects consistently rated Growing Squares above Hasse diagram with respect to efficiency, ease-of-use and enjoyment. The mean response values to the five point Likert-scale questions are summarized in Figure 8. The complete data analysis table is presented as Table 4.

The subjects' responses to the efficiency question (Q2, Table 4) showed a higher rating for the Growing Squares visualization than Hasse diagrams in both sparse (means 3.83 (s.d. .39) and 2.75 (s.d. .97)) and dense data densities (means 3.13 (s.d. .68) and 1.58 (s.d. .67)). Both higher rating readings were significant (Friedman Tests, $p = .0209$ for the sparse case and $p = .0039$ for the dense case). The subjects' response to the ease-of-use question (Q1, Table 4) also showed a higher rating for the Squares visualization in both sparse (means 3.92 (s.d. .67) and 2.67 (s.d. .89)) and dense data densities (means 2.79 (s.d. .78) and 1.46 (s.d. .66)). Both higher rating readings were significant (Friedman Tests, $p = .0094$ for the sparse case and $p = .0015$ for the dense case). The subjects' response to the enjoyment question (Q3, Table 4) also showed a higher rating for the Squares visualization in both sparse (means 3.92 (s.d. .79) and 3.00 (s.d. .43)), and dense data densities (means 3.25 (s.d. .85) and 1.92 (s.d. .67)). Both higher rating readings were significant (Friedman Tests, $p = .0094$ for the sparse case and $p = .0015$ for the dense case).

Figure 8 shows, not surprisingly, that the density of the data set strongly influenced the subjects' responses to each question for both visualizations. This difference is reliable for all but the enjoyability question (Friedman Tests). The subjects' response to this question (Q2, Table 4) when using the Growing Squares visualization shows a higher rating when small data sets are considered (means 3.92 (s.d. .79) for sparse sets and 3.25 (s.d. .75) for large sets), but on the other hand, this is not a significant difference ($p > .05$).

The final ranking questionnaire shows that most subjects preferred the Growing Squares technique over Hasse diagrams with regard to ease of use, efficiency, and enjoyment (Table 3). Overall, the results from this ranking are very

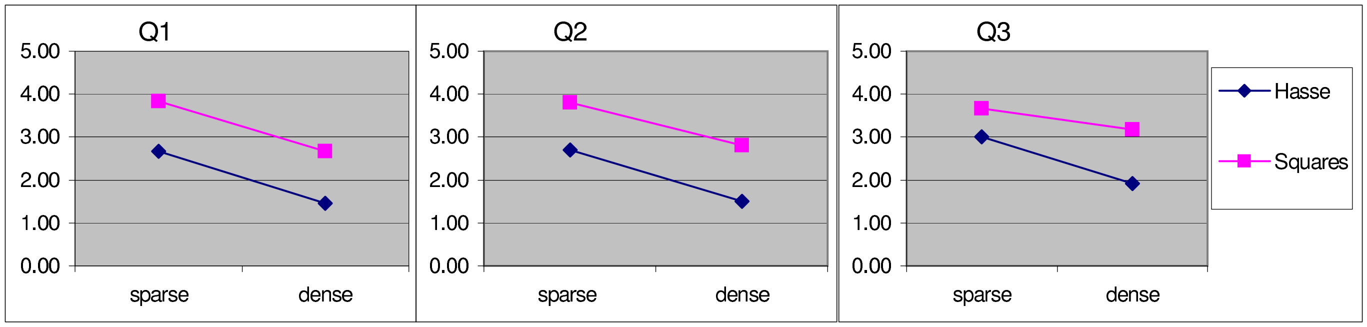


Figure 8: Responses to Q1-Q3 5-point Likert-scale questions across sparse and dense data densities for the Hasse and Growing Squares methods.

Question	Prefer GS?
Rank visualizations w.r.t. ease-of-use	92 %
Rank visualizations w.r.t. efficiency	83 %
Rank visualizations w.r.t. enjoyment	92 %

Table 3: Subject responses to ranking the two visualization.

favorable for the Growing Squares method.

7 Discussion

This work was motivated by our research effort to provide new techniques for visualizing causal relations that gives a clearer view of the information flow in a large distributed systems than traditional visualizations. Our technique is based on animation, colors and patterns in both 2D and 3D. A formative evaluation at the onset of the project and a user study at the end was conducted to validate our belief in the new visualization. To summarize, the results from the user study shows that the Growing Squares visualization technique is consistently faster and more efficient than Hasse diagrams. This difference, however, is not statistically significant for the general case, although it is significant for the sparse data set case.

The task completion times were consistently better for Growing Squares than Hasse diagrams, but there was a large variation on the times of different users. This came as a result of the different interpretation that subjects gave the definition of an overview task; some users correctly used the two methods with the intention of quickly gaining an overview of the system trace, while others spent much time trying to get a totally accurate answer. This problem is inherent in the nature of overview tasks, and is hard to avoid; we suspect, however, that we would have been able to prove significance with more test subjects.

On a related note, while our results failed to prove that our method scales better than Hasse diagrams, we firmly believe that Growing Squares does indeed perform better than traditional diagrams for large system traces. However, as Figure 7 suggests, this is a constant factor of improvement. For large systems some kind of hierarchical clustering scheme need to be used to group sets of processes into process groups. This technique would also benefit traditional classic Hasse diagrams.

It might be argued that users in general pursue accurate answers and not approximations. However, discussions with our focus group suggest that the role of causal relation visualization is to provide the user with a quick overview, and that accurate answers are best found using direct queries and offline analyses, not graphical diagrams.

Our main design goal with the Growing Squares method was to provide the user with an easily accessible map of the information flow and influences in a distributed system. Thus, the intention of the method was primarily for users to study the state of the system at the end of its execution, and to use the animation only when specifics about the dynamic execution was needed. Experiences from the user study showed that this was also the way the majority of the users employed the visualization, and that the animation aspect thus did not interfere with our objective of providing an overview of the system under study.

All test subjects were well-familiar with Hasse diagrams prior to carrying out the experiment whereas they knew nothing of the Growing Squares visualization, yet performed consistently better using the new technique in almost all cases. This, we think, suggests that the Growing Squares method is intuitive and easily accessible, and that the method with practice might become significantly more efficient than Hasse diagrams. The subjective ratings also support this belief. These clearly show that users prefer the Growing Squares visualization over Hasse diagrams. In addition, the enjoyability rating of the visualization did not decrease significantly as the data density increased.

The positive feedback that we have received from the subjects suggests that these kinds of alternate visualization methods of causal relations are indeed useful and worthwhile avenues for future research. By combining them with traditional methods such as Hasse diagrams, users will be able to use the strengths of different methods to solve different problems. In addition, the ability to view systems of causal relations from different perspectives will greatly aid in understanding the mechanics of the system.

8 Conclusions

The concept of causal relations is pervasive in human thinking and is a prime tool for determining the underlying principles of almost any system in most scientific fields. In the field of software engineering, causal relations reside in the core of the development process of parallel and distributed software systems. Visualization environments for parallel and distributed software typically offer animations of causal relations represented as Hasse diagrams (time-space diagrams). However, understanding complex causal relations is difficult, and traditional visualization techniques such as Hasse diagrams offer poor and not direct overview. We have developed

Question	Hasse diagrams		Growing Squares		Reliability	
	sparse	dense	sparse	dense	Hasse/GS	Density
Q1. Rate the visualization w.r.t. ease-of-use.	2.67 (.89)	1.46 (.66)	3.92 (.67)	2.79 (.78)	yes	yes
Q2. Rate the visualization w.r.t. efficiency.	2.75 (.97)	1.58 (.67)	3.83 (.39)	3.13 (.68)	yes	yes
Q3. Rate the visualization w.r.t. enjoyability.	3.00 (.43)	1.92 (.67)	3.92 (.79)	3.25 (.75)	yes	yes/no†

† Density does not significantly influence the enjoyability of the Growing Squares animation.

Table 4: Mean (standard deviation) responses to 5-point Likert-scale questions. Reliability is defined as being significant at the .05 level.

a new technique, called Growing Squares, that uses color coding and a metaphor based on pools of color growing over time to visualize causal relations in an intuitive fashion that provides quick and direct overview of the information flow in the system under study. User studies indicate that our technique is both faster and more enjoyable to use than Hasse diagrams.

9 Future Work

The Growing Squares method described in this paper has been used to analyze the execution of parallel and distributed systems, but the general nature of causal relations allows the technique to be used for other purposes as well, something which would be interesting to try. In addition, we are investigating extensions and improvements to the original scheme, including polishing of the prototype Growing Pyramids 3D extension, various layout algorithms, and adding the possibility to click and drag processes around in the visualization. Beyond these incremental improvements we are also looking into alternate techniques of visualizing causal relations, especially in 3D.

Acknowledgements

We would like to thank the researchers of Chalmers University of Technology who took part in the panel discussion at the beginning of the project. Thanks to David Modjeska for reviewing and commenting on the paper in its early stages. We also thank the test subjects who participated in the user study.

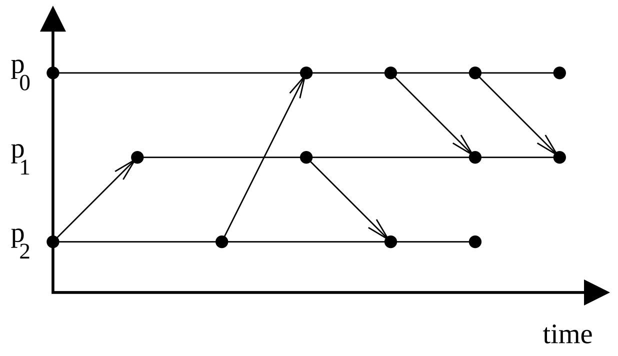


Figure 9: Hasse diagram visualization with 5 processes.

References

Alberts, D., Gutwenger, C., and Nher, P. M. S. 1997. AGD–library: A library of algorithms for graph drawing. In *Proceedings of the Workshop on Algorithm Engineering (WAE '97)*, 112–123.

Battista, G. D., Eades, P., Tamassia, R., and Tollis, I. G. 1999. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall.

Bederson, B. B., Hollan, J. D., Perlin, K., Meyer, J., Bacon, D., and Furnas, G. W. 1996. Pad++: A zoomable graphical sketchpad for exploring alternate interface physics. *Journal of Visual Languages and Computing 7*, 3–31.

Bederson, B. B., Meyer, J., and Good, L. 2000. Jazz: An extensible zoomable user interface graphics toolkit in Java. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2000)*, 171–180.

Bemmerl, T., and Braum, P. 1993. Visualization of message passing parallel programs with the TOPSYS parallel programming environment. *Journal of Parallel and Distributed Computing 18*, 2 (June), 118–128.

Cormen, T. H., Lesierson, C. E., Rivest, R. L., and Stein, C. 2001. *Introduction to Algorithms*, second ed. MIT Press, Sept.

Heath, M. T., and Etheridge, J. A. 1991. Visualizing the performance of parallel programs. *IEEE Software 8*, 5 (Sept.), 29–39.

Heath, M. T. 1990. Visual animation of parallel algorithms for matrix computations. In *Proceedings of the Fifth Distributed Memory Computing Conference*, 1213–1222.

Koldehofe, B., Papatriantafilou, M., and Tsigas, P. 1999. Distributed algorithms visualisation for educational purposes. In *Proceedings of the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITICSE-99)*, 103–106.

Kraemer, E., and Stasko, J. T. 1998. Creating an accurate portrayal of concurrent executions. *IEEE Concurrency 6*, 1 (Jan./Mar.), 36–46.

Lamport, L. 1978. Time, clocks and the ordering of events in distributed systems. *Communications of the ACM 21*, 7, 558–564.

Levkowitz, H., and Herman, G. T. 1992. Color scales for image data. *IEEE Computer Graphics and Applications 12*, 1 (Jan.), 72–80.

Levkowitz, H., Holub, R. A., Meyer, G. W., and Robertson, P. K. 1992. Color versus black and white in visualization. *IEEE Computer Graphics and Applications 12*, 4 (July), 20–22.

Moses, Y., Polunsky, Z., and Tal, A. 1998. Algorithm visualization for distributed environments. In *Proceedings IEEE Symposium on Information Visualization 1998*, IEEE, 71–78.

Perlin, K., and Fox, D. 1993. Pad: An alternative approach to the computer interface. In *Proceedings of Computer Graphics (SIGGRAPH 93)*, vol. 27, 57–64.

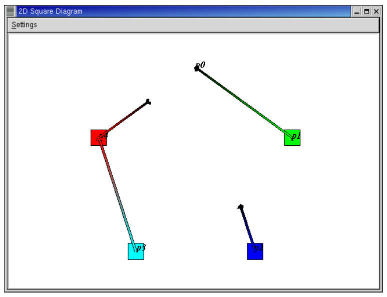
Socha, D., Bailey, M. L., and Notkin, D. 1989. Voyeur: Graphical views of parallel programs. In *Proceedings of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, ACM SIGPLAN Notices 24*, 206–215.

Stasko, J. T., and Kraemer, E. 1993. A methodology for building application-specific visualizations of parallel programs. *Journal of Parallel and Distributed Computing 18*, 2 (June), 258–264.

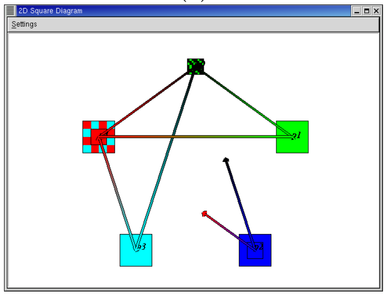
Topol, B., Stasko, J. T., and Sunderam, V. 1998. PVaniM: a tool for visualization in network computing environments. *Concurrency: Practice and Experience 10*, 14 (Dec.), 1197–1222.

Ware, C., Neufeld, E., and Bartram, L. 1999. Visualizing causal relations. In *Proceedings of IEEE Information Visualization 99*.

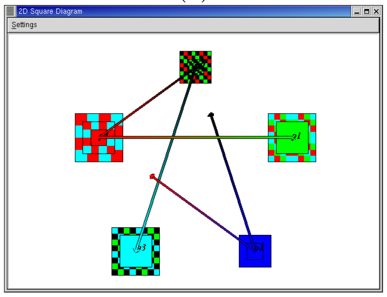
Wyszecki, G., and Stiles, W. S. 1991. *Color Science: Concepts and Methods, Quantitative Data and Formulae*, second ed. John Wiley & Sons.



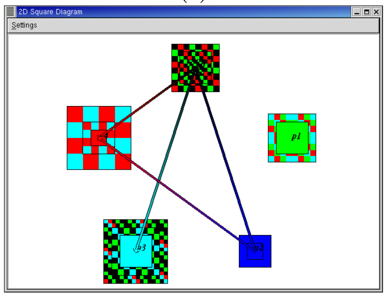
(a)



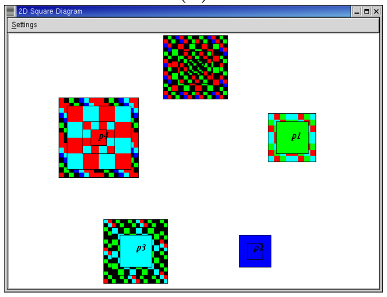
(b)



(c)

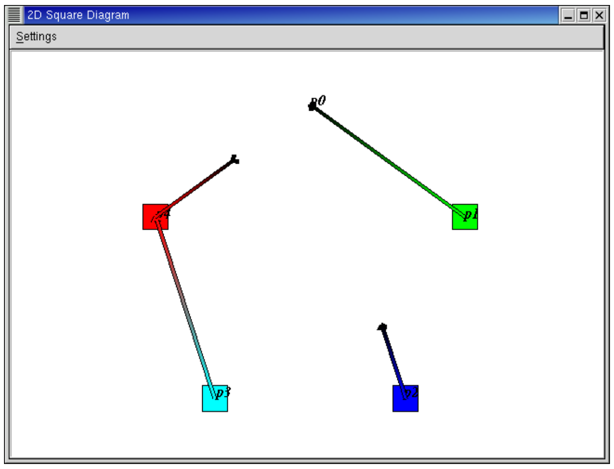


(d)

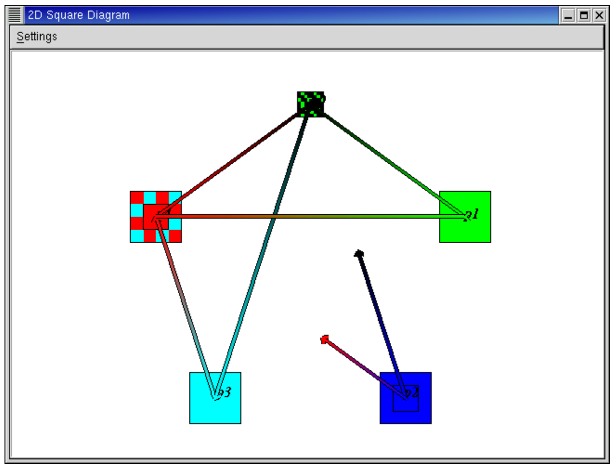


(e)

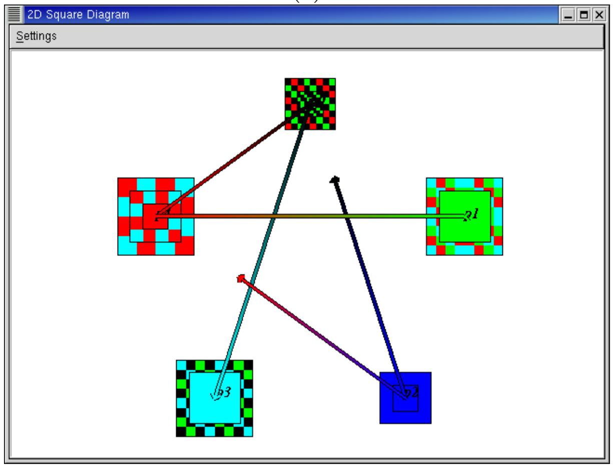
Figure 10: Growing Squares visualization of the dynamic execution of a 5-process distributed system.



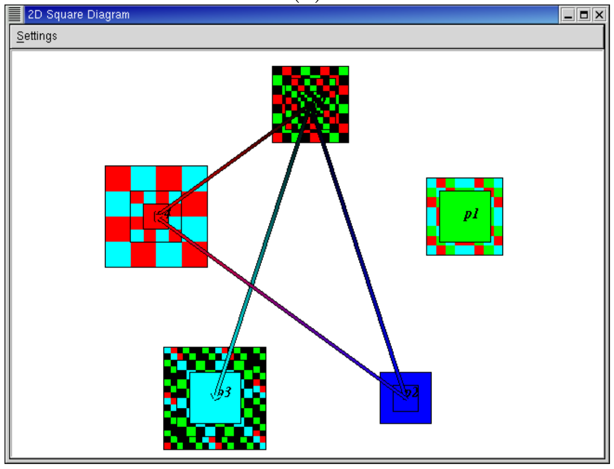
(a)



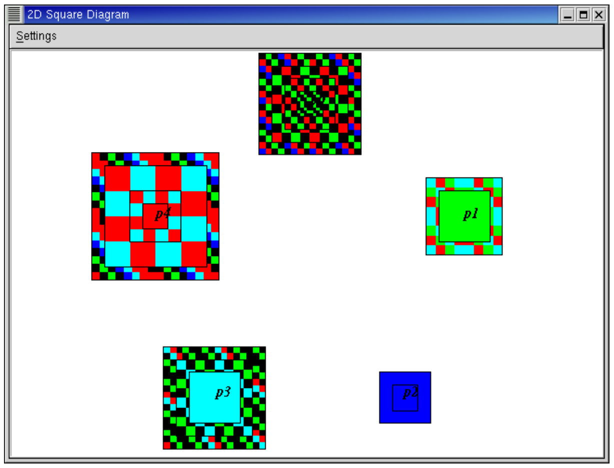
(b)



(c)



(d)



(e)

Color Plate: Growing Squares visualization of the dynamic execution of a 5-process distributed system.