Cluster Computation in Step With Real-Time Dance: An Experiment in Art/Science Collaboration

Jennifer Burg and Tim Miller
Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109

Abstract: This paper describes a collaboration among computer scientists, dancers, and musicians on a production entitled "Fibonacci and Phi." Thematically, the production explored mathematical concepts that have aesthetic appeal, capturing the ways in which mathematical beauty gives shape to nature and art, and expressing the human response to these forms. Technically, the production tested the limits of parallel cluster computation in real-time multimedia and performance art. The result was a dance performance that wove together science and art in a way intended to draw new audience members into both realms.

The History

The Winston-Salem Alban Elved Free Space dance project began with a question: How well can artists and scientists communicate, and what might they create if they try? The first Free Space production was a collaboration between Alban Elved Dance Company and Duke University scientists. The centerpiece of their 2001 performance was a ring of infrared cameras, called Argus, which sensed the dancers' motions and displayed them from multiple perspectives, adding novel and unsettling dimensions to the motions and space. (Argus was the one hundred-eyed monster of Greek mythology.)

In 2002, the Alban Elved Dance Company approached faculty of the Wake Forest University Department of Computer Science to explore the possibility of a similar collaboration. The result was "Fibonacci and Phi," a one and one half hour production

staged in December 2003 that explored the intriguing ubiquity of the sequence 1,1,2,3,5,8,11,… and the Golden Ratio, Phi, in the beauty of natural and artistic creations. Computer scientists produced digital images and movies, fractal-generating programs, 3-D and stereo animations, and even poetry, while musicians contributed digitally-produced music in intervals and cadences of Fibonacci and Phi – all to be woven together by Alban Elved choreographer Karola Luttringhaus.

The Inspiration

The Fibonacci sequence and the Golden Ratio, Phi, have been well-known to scientists and mathematicians since ancient times. While the Fibonacci sequence is deceivingly simple, this sequence is found in an amazing number of natural forms, ranging from the spiral of nautilus sea shells to the growth of leaves on a branch and seeds on the face of a sunflower. The sequence also is evident in the mathematical properties of fractals. Where the Fibonacci sequence appears, we generally find Phi, the Golden Ratio, as well. Phi's definition is also simple. If a line segment is divided into a smaller part A and a larger part B, and the ratio of the length of A to the length of B is the same as the ratio of the length of B to the whole segment, then this ratio is the Golden Ratio, Phi. Given its first explicit definition by Euclid more than 2000 years ago, Phi has been associated with the discovery of irrational numbers, the proportions of classic paintings and architecture, and the geometry of beautiful phenomena of nature.

The dance performance was woven together with a poem entitled "Phi," written by one of the computer scientists. (The poem is given at the end of the paper.) The reading of the poem was divided among the dance pieces of the performance, one or two stanzas at a time per dance, with digital imagery as a backdrop on stage. Thematically,

the performance explores the human need to decipher the cryptic messages of Fibonacci and Phi – to "say and count" the beauty around us, putting it into words and analyzing it with pure mathematics, thereby helping us to understand the beauty and make it our own.

Mise-en-Scène

During the dance numbers of "Fibonacci and Phi," a variety of digital images were projected on two screens – one standard white screen 40 × 25 feet at the back of the stage, and the other a translucent black scrim in the foreground. The images included an opening montage reflecting the myriad visions of our waking and sleeping lives; 3-D animated mannequins controlled in real-time by dancers via joystick and mimicked by other dancers on stage; a fractal tree that grew leaves in the form of words and phrases, shedding these leaves to create a poem "on the fly"; digital movies and images of fractals designed to suggest a sunrise in the opening number and a night sky "galaxy" fractal in closing; and a real-time animation of a Mandelbrot fractal through which dancers move in a playful and competitive duet. This paper focuses on the technical challenges of the real-time fractal animation. (For a more detailed description of the entire production, see "Dancing with Fractals and Antique Snowflakes Through the Magic of Fibonacci and Phi" at http://www.cs.wfu.edu/~burg/papers/DancingWithFractals.pdf.)

The Problem

For the "Fibonacci and Phi" production, we decided to take on the challenge of animating a Mandelbrot fractal in real-time in response to the movements of dancers. This entailed designing a way for the dancers to communicate with the fractal-generating program and parallelizing the fractal generation to keep up with the speed of the dancers.

While parallel implementations of fractal generation abound, we know of no other system that does fractal computation coordinated with dancers' movements in a real-time performance.

The first step in the design was the communication interface between the dancers and the fractal computation program. In previous productions, the Alban Elved Dance Company had used a movement-to-MIDI converter to generate MIDI sounds in response to the dancers' motion. The movement-to-MIDI converter works as follows: Laser beams are aimed across the stage such that when the dancers move through them, breaking the beams, a MIDI signal is generated. Our innovation to this system was to capture the signal and, rather than generate a MIDI sound, send it to a computer as a message to recompute the Mandelbrot fractal. Thus, the movement-to-MIDI device was directly connected to a computer located backstage.

Early experiments made it clear that sequential computation of the Mandelbrot fractal would not be sufficiently fast for real-time animation, so the second step in the design was to implement a parallel solution. The choreography was to have dancers moving through the laser beams, each time signaling that the fractal should be recomputed so that it looked just slightly closer than the previous frame. How many times per second would the fractal need to be redisplayed and recomputed in order to create the effect of moving closer into the infinitely self-replicating structure, always with the same perfect 1024 × 768 full screen resolution? For smooth animation, each step had to be small – only a few pixels closer in each dimension. For fast movement, with each step so small, the fractal would have to be recomputed and redisplayed several times per second.

We decided to use an MPI parallel implementation of the Mandelbrot computation on a Linux cluster. The network connectivity between the stage and Linux cluster is pictured in Figure 1. The stage is pictured in Figure 2.

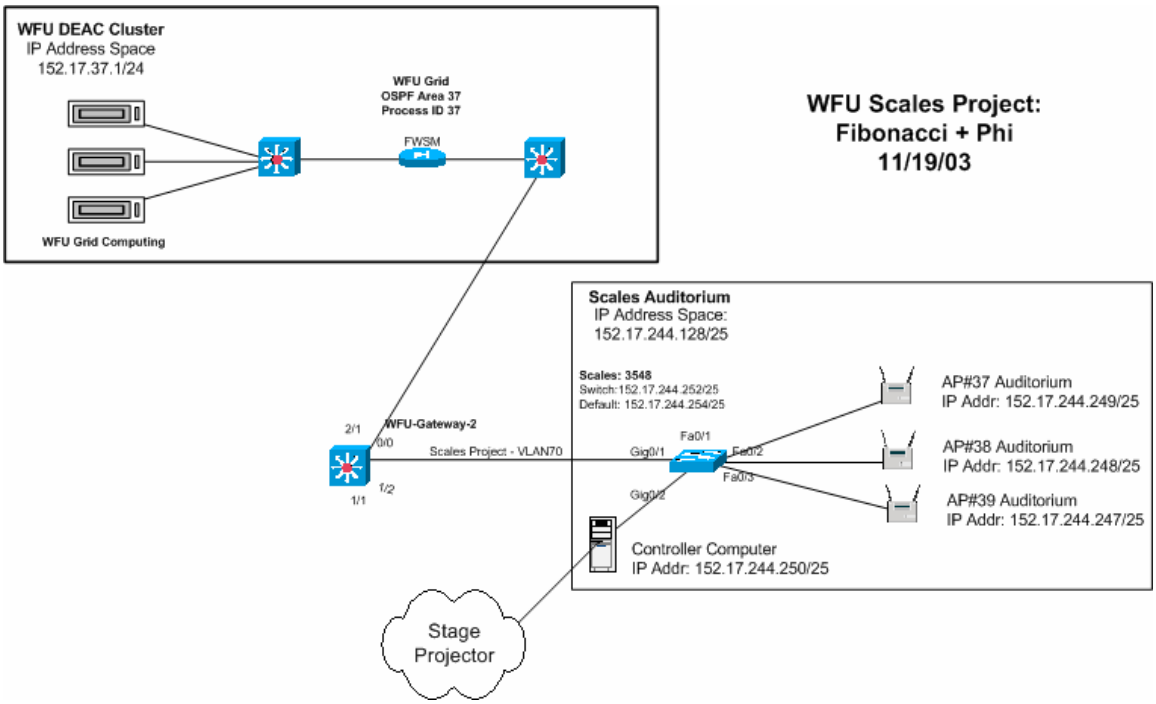


Figure 1. Network and Hardware Setup of Fractal Computation System

Figure 2. Fractal Duet in "Fibonacci and Phi."
(A still clip extracted from a digital video shot by
Ching-Wan Yip, Wake Forest University.)

When the dancers move through one of six laser beams, a signal is sent to the movement-to-MIDI converter, which forwards the signal to a computer backstage. Depending on which beam is broken, the signal could indicate that the fractal should be recomputed *in*, *out*, *up and in*, *down and in*, *left and in*, or *right and in*, where *in* and *out* mean *on a smaller scale* and *on a larger scale*, respectively. Moving *in* gives the audience the illusion of driving into the fractal. The computer runs a client program that forwards the message to the master process of the Linux cluster. (The computer on stage is called the *client computer*.) The master process divides the work of computing the fractal's pixel colors among *n* dedicated processors. Using X Windows functions, the master process sends the data back to the stage, which is connected to a projector that displays the fractal at a resolution of 1024 × 768 on a screen at the back of the stage that is 40 feet × 25 feet.

Based on this set up, preliminary analysis identified three possible bottlenecks in the animation. Table 1 lists these bottlenecks and strategies for speeding up the computation and communication.

Possible Bottleneck	**Possible Solutions**
computation of pixel colors	parallel implementation on Linux cluster
communication among processes of the cluster	• myrinet interprocess network communication (as opposed to ethernet) • run-length encoding of pixel data
communication of pixel data from master process to computer on stage	gigabit ethernet connection from client computer to Linux cluster

Table 1. Possible Bottlenecks and Solutions

The problem was to speed up the computation of each frame to the point that navigation through the fractal would be, from the audience's point of view, both smooth and sufficiently fast to be interesting. The visual speed of the navigation is a product of how quickly a section of the fractal appears to grow larger or "get closer." The

smoothness of the navigation is a product of how much difference there is in the size of a given area of the fractal from one frame to the next. A smoother navigation has less difference between one frame and the next, but then to make the “zoom in” sufficiently fast, it is necessary to redisplay frames at a high rate. Subjective evaluations were made of the properties of “speed of navigation” and “smoothness.”

More objectively, the problem was to find ways to speed up each of the potential bottlenecks, to measure the extent to which our programming and network changes in fact improved performance, to determine if one of the bottlenecks dominated the others, and ultimately to animate the fractal at the optimum rate for what was subjectively determined to be fast, smooth animation.

The fractal computation program is based on the well-known simple iterative equation

$$f(z) = z^2 + c$$

where c and z are complex numbers. To compute a pixel’s color, c is initially the pixel's position and z is 0, and $f(z) = z^2 + c$ is computed repeatedly for a maximum number of iterations or until the value converges. The result is a "classic" Mandelbrot fractal like the one shown in Figure 2. This is the fractal used in the “Fibonacci and Phi” fractal duet.

Variations of the Mandelbrot fractal, called Julia fractals, can be created by mapping the initial z to the pixel’s position and appropriately selecting c to remain constant for each pixel. Figure 3 shows a Julia fractal we designed for the closing “Night Sky” piece of the dance performance. The design was created by finding a constant c that produced a galaxy-like shape, creating a suitable color map, and painting random

points of light at densities varying by color map to soften the edges and created a twinkling star effect as repeated computations zoom in on the fractal.

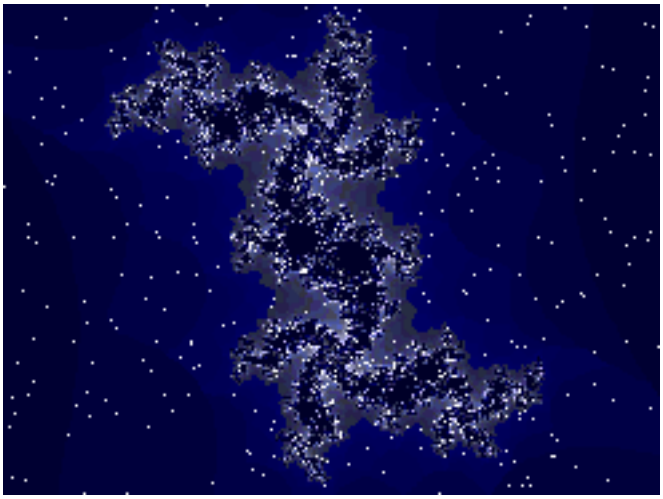


Figure 3. Julia Fractal Designed to Resemble a Galaxy in a Night Sky

In the discussion that follows, one image of a fractal will be called a *frame*. The computation of a frame is obviously a challenge for real-time animation. Let $x \times y$ be the resolution of the fractal and let *t* be the maximum number of iterations to compute one pixel. Then the worst-case complexity for computing one frame is

$$x * y * t$$

which for our application is

$$1024 * 768 * 1000 = 786{,}432{,}000$$

The complex number computation is done as follows: *z* is given by (z_r, z_i) and *c* is given by (c_r, c_i) as described above. Let *z_new* be the value of *z* that will become the input to the subsequent computation, with components z_new_r and z_new_i. Consider first what $z^2 + c$ yields.

$$z\_new = z^2 + c =$$
$$(z_r + z_i * i)^2 + (c_r + c_i * i) =$$
$$(z_r{}^2 + (2 * z_r * z_i * i) + (z_i{}^2 * i^2)) + (c_r + c_i * i) =$$
$$z_r{}^2 + (2 * z_r * z_i * i) - z_i{}^2 + c_r + (c_i * i) =$$
$$z_r{}^2 - z_i{}^2 + c_r + \left[(2 * z_r * z_i * i) + (c_i * i)\right]$$

To do this computation using only real number values (with $\sqrt{-1}$ implicit in the imaginary component of each complex number), we need to separate out the real from the imaginary components (the ones with a factor of i), as we have done in the last line above. This gives

$$z\_new_r = z_r{}^2 - z_i{}^2 + c_r =$$
$$z_r * z_r - (z_i * z_i) + c_r$$

and

$$z\_new_i = 2 * z_r * z_i + c_i$$

Thus, each pixel computation requires 4 multiplications and 3 additions. Clearly, not all pixels will require the maximum number of iterations – only those that are painted black. The frames with the most black are the most expensive to compute.

The second potential bottleneck is the interprocess communication among cluster processes, a function of the amount of data to be sent. Communication from the master to the slaves is trivial, involving a small amount of data. The master tells each slave the range of values on the complex number plane to which the pixel values are to be mapped. Initially, the range is -3.5 to 0.5 in the x direction and -1.5 to 1.5 in the y direction, but as we zoom in on the fractal this range shrinks, yielding closer and closer views. Communicating the dimensions of the complex number plane requires only four values – beginning and ending positions in both the x and y directions.

Communication from the slaves to the master requires more data than communication from the master to the slaves – $1024 * 768 * b$ bytes of data, where b is the number of bytes per pixel. Without run-length encoding, this amount of data would be the same for all frames.

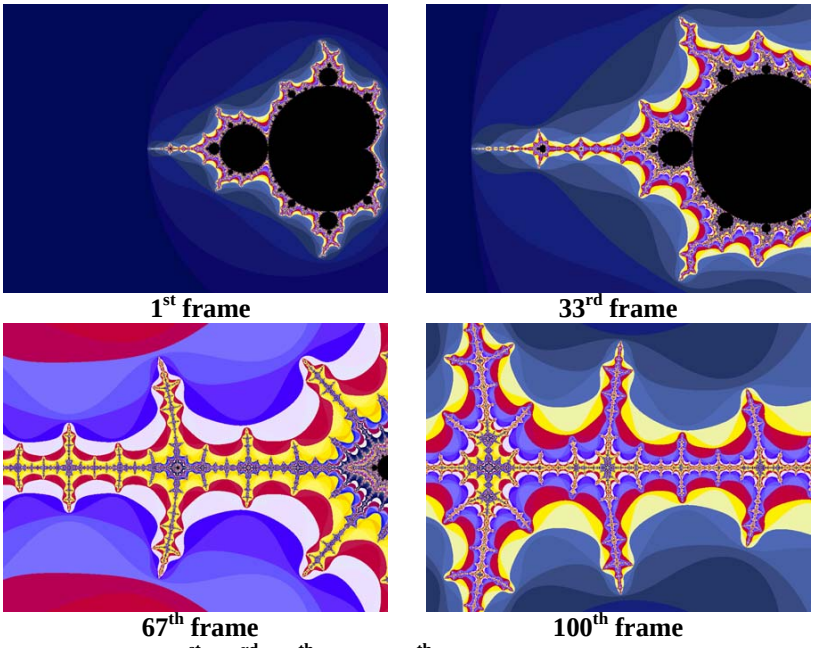
To make the situation worse, the master process itself becomes a bottleneck, as all pixel data must reach the master process before the X Windows call is made for displaying the frame. In an early version of the program, slave processes each wrote their pixel data directly to the X Windows server on the client computer. However, this resulted in rows that came streaming across the screen one at a time. For smooth redisplay, the program was rewritten so that the master stores all the pixels into a pixmap in memory, and then calls for the redisplay of the entire frame with one X Windows call. The "dead time" of the master process waiting for all slaves to report in with pixel data thus became a factor.

The third potential bottleneck is the communication from the master process to the computer on stage. This communication is done through repeated X Windows calls to set the color for a pixel or pixels as the pixmap is written, and a single call to XCopyArea to write each frame to the X Window after it has been constructed.

Experiments

The first experiments were to determine, empirically, the average complexity of computing a frame and the average amount of data sent from the slaves to the master, comparing these values to the worst case. For these experiments, the same 100 frames were computed each time, beginning with the Mandelbrot fractal over the complex number plane in the range $[(-3.5,0.5),(-1.5,1.5)]$. Zoom-in was repeated for 100 frames,

each time as if a pixel area had been selected that was two pixels smaller in each dimension (moving in one pixel on left, right, top, and bottom). For example, in the first step, this would entail zooming in on the pixel area [1, 1022] in the x direction and [1,766] in the y direction. (Every fourth frame, the y direction was not changed in order to maintain the 4:3 ratio of the frame.) The new pixel range was mapped to the complex number plane, and this area was then recomputed and redisplayed at the original resolution – 1024 × 768 – resulting in a closer view of the area with no loss of resolution detail. The 1st, 33rd, 67th, and 100th frames of the computation are shown in Figure 4.



1st frame **33rd frame**

67th frame **100th frame**

Figure 4. 1st, 33rd, 67th, and 100th frame of Mandelbrot fractal

With the 100 frames fixed for the experiments, the total number of multiplications and additions remains constant, independent of the number of cluster processes and variations in network technology.

The average, minimum, and maximum number of iterations for the 100 frames is given in Table 2. (If i is the actual number of iterations, then the number of

multiplications and additions would be 4*i and 3*i, respectively, as discussed above.) The table shows that on average, only about 8% of the maximum possible number of iterations had to be performed to compute a frame.

maximum # of iterations possible per frame (case where all pixels are black)	**average # of iterations over all 100 frames we computed**	**minimum # of iterations over all 100 frames we computed**	**maximum # of iterations over all 100 frames we computed**
786,432,000	65,468,128	10,799,199	133,629,685

Table 2. Maximum, minimum, and average number of iterations performed in computing the pixel colors of the Mandelbrot fractal

In the next refinement, run-length encoding was implemented to reduce the amount of pixel data sent between the slaves and the master. Without run-length encoding, the amount of data would be a constant $1024 * 768 * b$ per frame. Run-length encoding consists of sending a two byte integer *d* and a two-byte color code to indicate *d* consecutive bytes of the same color, as opposed to sending *d* two-byte color codes. Given the nature of the fractal images, which have long runs of the same color pixels, RLE reduces the data communication significantly. Table 3 shows the maximum, minimum, and average number of bytes sent per frame over all 100 frames.

100 Mbit/second Network Connection from Client to Cluster						**1000 Mbit/second (GigE) Network Connection from Client to Cluster**					
ethernet connectivity in cluster			**myrinet connectivity in cluster**			**ethernet connectivity in cluster**			**myrinet connectivity in cluster**		
4 procs	**av.**	37650	**4 procs**	**av.**	37650	**4 procs**	**av.**	37650	**4 procs**	**av.**	37650
	min	1282		**min**	19966		**min**	3978		**min**	21599
	max	204812		**max**	91815		**max**	93990		**max**	89545
8 procs	**av.**	18842	**8 procs**	**av.**	18842	**8 procs**	**av.**	18842	**8 procs**	**av.**	18842
	min	294		**min**	6384		**min**	458		**min**	7378
	max	198182		**max**	41645		**max**	84269		**max**	40228
16 procs	**av.**	9435	**16 procs**	**av.**	9435	**16 procs**	**av.**	9435	**16 procs**	**av.**	9435
	min	86		**min**	3818		**min**	146		**min**	3188
	max	192357		**max**	19653		**max**	89121		**max**	20414

Table 3. Average, minimum, and maximum number of bytes per frame communicated per slave to the master over all 100 frames

From these data, it can be seen that run-length encoding reduces the amount of data transmitted between the slaves and the master per frame by approximately 90%. (For

example, with 4 processors, ethernet within the cluster, and Mbit/sec between client and cluster, the average data per frame is $37{,}650 * 4 = 150{,}600$ bytes as opposed to the $1024 * 768 * 2 = 1{,}572{,}864$ bytes per frame required with no encoding.)

For a baseline, the average time needed to compute one frame using a sequential program and a 100 Mbit network connection from the client computer to the cluster was measured. The result was that it took between 14 and 17 seconds to compute a frame, the time varying with network usage. This was far from the goal of real-time animation.

Pressured with an impending opening night for "Fibonacci and Phi," we gave the problem all the network speed and computational power we had available, including a gigabit ethernet connection from the client computer to the Linux cluster, 16 processors on the cluster for the fractal computation, run-length encoding of pixel data sent between the slaves and master, and myrinet connectivity in the cluster. The result was a smoothly-animated fractal that could be navigated by the dancers in real-time, with the last trigger in the "fractal duet" signaling a fast zoom-in into a black hole of the fractal.

In retrospect, after the show we were interested in analyzing exactly which part of the final system resulted in the biggest benefit. Was the gigabit ethernet connection all the way from the client computer to the cluster really needed? Was myrinet connectivity within the cluster actually necessary? How many processors were optimum? Would fewer than 16 processors actually be better because of the bottleneck funneling the X Windows data through the master process? These questions were important in light of the fact that a road show was being planned for "Fibonacci and Phi." If myrinet connectivity wasn't crucial on the cluster, it would be easier to build a portable cluster to take on the road, and fewer processors would be easier as well.

	4 processors		8 processors		16 processors	
• 100 Mb/sec from client to cluster • ethernet connectivity on the cluster • slower graphics card	average	0.558267	Average	0.477561	average	0.467278
	minimum	0.433044	Minimum	0.383174	minimum	0.360396
	maximum	0.959640	Maximum	0.639695	maximum	0.630410
• 100 Mb/sec from client to cluster • myrinet connectivity on the cluster • faster graphics card	average	0.551303	Average	0.466858	average	0.468283
	minimum	0.431126	Minimum	0.361114	minimum	0.359247
	maximum	0.779139	Maximum	0.600989	maximum	0.599170
• 1 gigabit/sec from client to cluster • ethernet connectivity on the cluster faster graphics card	average	0.303111	Average	0.178108	average	0.128352
	minimum	0.102596	minimum	0.096622	minimum	0.098177
	maximum	0.580423	maximum	0.439282	maximum	0.325958
• 1 gigabit/sec from client to cluster • myrinet connectivity on the cluster • faster graphics card	average	0.303086	Average	0.171172	average	0.137445
	minimum	0.100156	minimum	0.102299	minimum	0.106616
	maximum	0.624767	maximum	0.345077	maximum	0.325387

Table 4. Average, minimum, and maximum time to compute a frame for all 100 frames with varying cluster and client/server connectivity, varying number of processors on Linux cluster

Table 4 shows the average amount of computational time needed to compute a frame using 4, 8, and 16 processors; myrinet or ethernet connectivity on the Linux cluster; and either gigabit/sec or 100 Mb/sec ethernet between the client computer and the Linux cluster. These experiments show that with 16 processors, gigabit ethernet, and

myrinet, the fractal can be refreshed between 3 and 10 times a second. This was certainly enough for smooth, fast navigation through the fractal. It is interesting to note, however, that even less expensive resources may give sufficient power for the "fractal dance." With only 8 processors and ethernet within the cluster, the fractal can be animated at a rate of 2 to 3 times per second, fast and smooth enough for an interesting visual effect in the dance. These experimental figures indicate that it is possible to take the "Fibonacci and Phi" performance on the road with a smaller-scale, less expensive portable cluster.

Table 5 shows the extent to which the transmission of pixel data saturates the network between the client computer and the cluster. For 16 processors, an average data rate of 92 Mbit/sec using a 100 Mbit network connection as opposed to 235 Mbit/sec using gigabit ethernet indicates that gigabit ethernet speed between the client and the cluster makes a significant difference in the refresh rate of the animated fractal.

100 Mbit/second Network Connection from Client to Cluster				**1000 Mbit/second (GigE) Network Connection from Client to Cluster**			
ethernet connectivity in cluster		**myrinet connectivity in cluster**		**ethernet connectivity in cluster**		**myrinet connectivity in cluster**	
4 procs	80 Mbit/sec	**4 procs**	82 Mbit/sec	**4 procs**	146 Mbit/sec	**4 procs**	146 Mbit/sec
8 procs	92 Mbit/sec	**8 procs**	94 Mbit/sec	**8 procs**	235 Mbit/sec	**8 procs**	236 Mbit/sec
16 procs	92 Mbit/sec	**16 procs**	94 Mbit/sec	**16 procs**	235 Mbit/sec	**16 procs**	304 Mbit/sec

Table 5. Network Usage from Client To Linux Cluster

The benefit of myrinet connectivity within the cluster is also visible in Table 5, particularly in the difference between ethernet and myrinet for 16 processors. The data rate between the client and the cluster is 235 Mbit/sec when the cluster has an internal ethernet connection, whereas it is 304 Mbit/sec when the cluster has an internal myrinet connection. It may not be immediately clear why the data rate between the client computer and the cluster would be higher when the cluster, internally, has myrinet rather

than ethernet connectivity. Table 3 and Figures 5 and 6 help to explain this behavior. One benefit of myrinet appears to be that it distributes the work load better among the processors, as indicated in the difference between the average, minimum, and maximum number of bytes sent from any slave process to the master. There is less difference between the minimum and the average number of bytes transmitted per process for myrinet as opposed to ethernet. (See Table 3.) Figures 5 and 6 show the interprocess communication graphically, comparing the situation for myrinet (the top half of each figure) and ethernet (the bottom half) in the case of 4 processors. Figure 5 represents the interprocess communication for the first fractal frame. Figure 6 represents a later fractal frame.

Consider Figure 5. In the top half of the figure, each row represents one of the 4 processors, the top one being the master. Dark green sections indicate the process is "probing" or waiting for messages to be received. Light green sections show the process receiving a message from another process. Blue sections indicate the process is sending a message to another process. The arrows in the figure depict the direction of the communication and connect the same send and receive operation. According to the way the algorithm is written, the master gives 2 slices of the fractal frame to a processor, and then moves on to allocate the next 2 slices to any free processor. The algorithm looks for free processes in order of process number, so lower numbered processes have a preference for getting more work. With the slower ethernet connection, interprocess communication takes longer, and thus there is a greater chance that process 1 will have finished its fractal computation by the time that process 0, the master, is ready to send another slice. With myrinet, computation and communication time achieve a better

balance, and thus higher-numbered processes get work more often than is the case in the ethernet cluster. Figures 5 and 6 show this behavior in that the dark green blocks – wait blocks – are longer for higher numbered processes in the ethernet case than in the myrinet case.

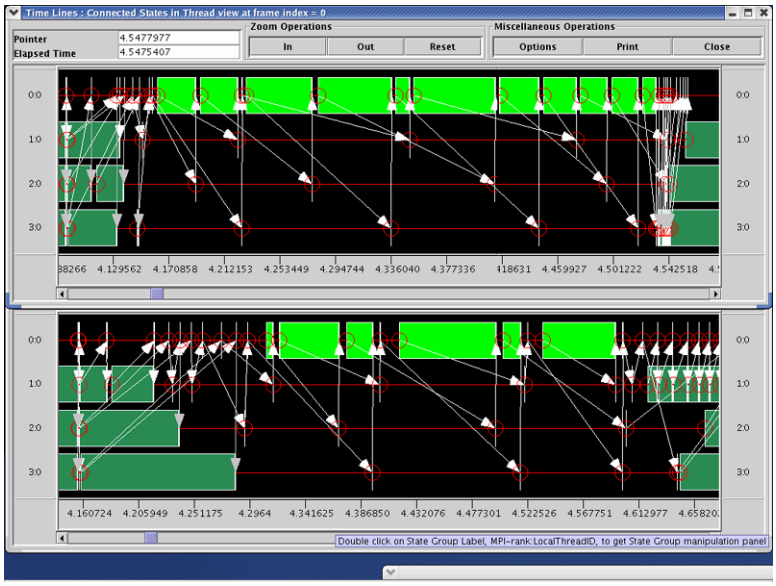


Figure 5. Interprocess communication for 4 processors for first fractal frame.
Top half of figure was taken during a run with myrinet in cluster.
Bottom half of figure was same computation on a run with ethernet in cluster.

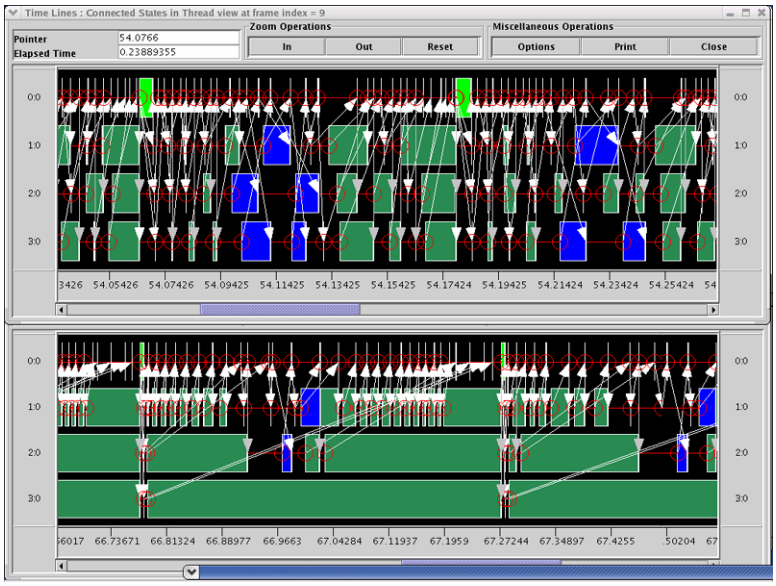


Figure 6. Interprocess communication for 4 processors for a fractal frame after the first one.
Top half of figure was taken during a run with myrinet connection.
Bottom half of figure was same computation on a run with ethernet connection.

Conclusions

"Fibonacci and Phi" fulfilled multiple purposes, depending on the vantage points of audience members, artists, and scientists who participated in the production or enjoyed the performance. The audience was introduced to the concepts of the Fibonacci sequence and the Golden Ratio through visual imagery accompanied by narrative and poetic interpretation. The audience members were also made aware of the computational power of parallel, fast-network computation through an interaction of dance and fractal imagery. The computational analysis reflected in this paper serves as a case study for high performance computing that the authors will be able to use in future parallel computation courses – an example that will quickly interest students and will introduce concepts of high performance computing in an engaging, graphical medium. The analysis is useful to the authors in helping them plan future dance/digital media/high performance computation collaborations, including the proposed road-show of "Fibonacci and Phi" using a portable 8-processor cluster with ethernet connectivity. The authors intend this work to be the first in a series interdisciplinary works that bring art to the scientifically-minded, and science and math to artists and humanists.

References

Briggs, J. (1992). *Fractals: The Patterns of Chaos.* New York: Touchstone Book/Simon & Schuster.

Livio, M. (2002). *The Golden Ratio: The Story of Phi, the World's Most Astonishing Number*. New York, Broadway Books.

Mandelbrot, B. B. (1988). *Fractal Geometry of Nature.* New York, W. H. Freeman and Co.

Pickover, C. (1995). *Keys to Infinity.* New York, John Wiley & Sons.

Phi

What are these haunting messages
Coded in cryptic languages
Through sights and sounds and senses
Speaking to us without words?

And how do we decipher
The beauty as it strikes us,
By saying it or counting it
Make it finally our own?

Who hears the golden music best,
Who sees with clearest vision?
And can they tell me what they see
And write down every note?

A child has eyes still bright and true
An ear open to voices.
A child hears shouts of tiny "Whos"
On dust specks in the air.

A child knows worlds hold worlds inside
Each world leads to another
And angels dance on heads of pins
When you reach infinity.

But children don't have words for this
And insufficient numbers
And as we age we want to say
Or count what we have known.

The ancients mathematicians sought
A language most eternal
And found in pure proportions
A number timeless and divine.

The golden ratio they called it
And Phi in Greek we named it.
And everywhere we find its mark
In nature and in art.

I see a message etched
In a ragged rocky coastline
And the pattern is repeated
In the ripples at my feet.

A fern unfurls its growing leaves
Like nature's own fresh fractal
So I paint a fractal of my own
To find the world inside.

I try to read a message
In the face of a sunflower
But I'm blinded by the spirals
Spinning left, and spinning right.

The spirals leave my dazzled grasp
A galaxy is born
And sends to me through heaven's time
A metaphor of stars.

A message whispers softly
In the angles of a seashell
And calls my soul to trace a curve
Down paths that never end.

I am told that these mute messages
All have Phi locked within them.
What is this magic number?
And what secrets does it hold?

We cannot write the number,
So irrational by nature.
Never ending, always changing
As it steps toward the sublime.

What would happen if we could
Know the endless perfection?
Say in words and in numbers
What we don't yet understand?

If we mark the musical intervals
With infinite precision
Can we make a human symphony
From the harmony of the spheres?

If we trace the seashell's spiral
Down endless perfect angles
Will we finally find the center
And meet the eye of God?

We cannot take the measure
Of Your exquisite beauty
Though it's woven in the fabric
Of our world and our flesh.

We cannot say Your name
Though it's written in the sky
Still we silently rejoice to read
The messages You send.