INTERACTIVE HIGH PERFORMANCE COMPUTING FOR MUSIC

Benjamin D. Smith
University of Illinois at Urbana-Champaign
School of Music

Guy E. Garnett
University of Illinois at Urbana-Champaign
eDream, Illinois Informatics Institute

ABSTRACT

The origins of computer music are closely tied to the development of the first high-performance computers associated with major academic and research institutions. These institutions have continued to build extremely powerful computers, now containing thousands of CPUs with incredible processing power. Their precursors were typically designed to operate in non-real time, "batch" mode, and that tradition has remained a dominant paradigm for high performance computing. We describe experimental research in developing the interactive use of a modern high-performance machine, the Abe supercomputer at the National Center for Supercomputing Applications on the University of Illinois at Urbana-Champaign campus, for real-time musical and artistic purposes. We describe the requirements, development, problems, and observations from this project.

1. INTRODUCTION

Computer musicians have gradually moved away from relationships with large-scale computing centers as personal computing power has increased and become more available. Personal computing has brought benefits such as interactivity (the ability to receive a response from the computer more rapidly than the human discriminative capability can discern, sometimes called "real time"), portability (enabling musicians and performers to bring their machines out of studios and onto stages and other venues), and freedom to develop and test software on flexible, individual schedules (allowing one to utilize the full capability of the machine at any time of day or night and for extended periods of time). However the rate of development in individual processor speeds has begun to slow, causing many manufacturers to move to multi-processor architecture. The world of High Performance Computing (HPC), continues to develop rapidly by incorporating CPUs in greater and greater number with incredible interconnect speeds. As of November 2010 there are approximately ten computers worldwide capable of at least theoretic peak performance in the petaFLOPS range (10^{15} sustained floating point operations per second)[21] with more poised to come online. Predictions anticipate exaFLOPS (10^{18}) machines by 2019 [20]. These machines, and their recent predecessors, consist of thousands of CPUs, connected with high-bandwidth buses and fiber-optic LANs. Typically, data processing jobs are run on these machines in a queued batch mode, wherein a user can submit a huge amount of data and algorithms, and receive notification when their job is complete (usually hours or days later).

Interactive HPC, a rapidly developing area of computing [15], allows for more flexible usage and holds out promises of real-time use of supercomputers through dynamic scheduling. This allows users to obtain the use of portions (or all) of the computer for interactive, though still not typically real-time, program execution. The interaction rate is still fairly long, taking the form of loading presets or setting initial parameters and then simply observing the process. We desired to push this capability further, enabling near instant, human control of a complex simulation using a multi-dimensional controller, i.e. an acoustic violin. We developed and tested a project to explore this potential, creating a real-time computer music instrument employing high definition video and audio displays. Through the National Center for Supercomputing Applications (NCSA) on the University of Illinois at Urbana-Champaign (UIUC) campus we obtained access to Abe, a supercomputer containing 9600 Intel 64 CPUs capable of reaching 89.47 teraFLOPS (representing the peak of supercomputing performance when it came online, until being surpassed in the last decade), which formed the computational core of this project.

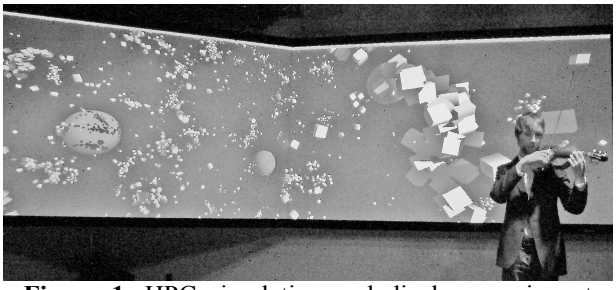

Figure 1. HPC simulation and display running at NCSA, January 2011 (Benjamin Smith, violin).

2. MOTIVATION

The model of the "instrument" is commonly used in computer music development to frame interactive systems and their function in performance [4, 22]. This notion provides a well-prescribed formula for interaction design and evaluation, including action-effect expectations (usually, but not always, one action to one sonic event), immediacy of response, and ease of use while maintaining potential for virtuosity. Cases of such computer music instruments employing personal computers abound, and while the computing power available in commercially available technology today is impressive, it remains limited. With the computing power of current generation HPC systems it becomes

possible to consider experiments well beyond the purview of personal computing, such as a real-time molecular simulation and sonification of an acoustic instrument, or video and audio displays of huge, continuously updating embedded, sensor arrays.

The realization of a true supercomputer instrument requires continuing development of several components: high resolution gestural controllers and input devices; strong, controllable models and simulations; and high definition audio and video displays capable of rendering massive amounts of data in real-time. The Allosphere [9] is one such display environment, capable of rendering over a 360° field of view, with full surround sound. However, the current applications running on the Allosphere, as is typical of many visualization applications, are apparently driven by a single desktop computer, convolving the interaction, simulation, and, sometimes, display into one process. Segmenting these components and giving each an independent calculation rate allows each to take full advantage of the processing power available [7]. This can be especially important when a given segment (such as a simulation) can benefit from a finer grain time-step delta, but another part (such as the display rendering) needs to operate at a significantly slower, or variable rate.

Historically, due to the batch mode orientation of most HPC systems, supercomputer music applications took the form of large-scale algorithmic composition and synthesis programs [8, 10, 11]. While the simulations most commonly run in HPC environments are of a scientific nature, seeking to better model and understand some real observed phenomena [24], the format also lends itself well to large scale interactive creative simulations, effectively becoming a computation intensive algorithmic composition systems. We take this view of the simulation running on Abe: it becomes an advanced compositional algorithm with many interacting parts, all globally steered by a live musician.

In the field of supercomputing, Interactive HPC refers to real-time operation, but does not usually entail a closed input-to-computation-to-display loop. Even complex visualization systems [19] typically depend solely on initial inputs, limiting the interaction with the simulation to start, pause, and stop commands. They are thus aimed at speeding up the development cycle of the programmer, or the exploration cycle of the scientist, rather than enabling a performance oriented interaction. Creating a fully interactive computing environment, in our view, entails real-time control at a detailed level during the running of the simulation, providing the ability to steer the course of the simulation during execution [24], which requires at least a simulation with all computations relevant to a given time-step occurring at or before the required output of that time-step. Our solution is to involve an expert practitioner of an acoustic musical instrument, the violin, providing them with multi-dimensional control of the simulation through the analysis of their sound. In effect, the violin becomes a highly dimensioned interface to a larger digital instrument involving the HPC simulation, and graphical and aural displays.

3. DESIGN

The requirements of the system we sought to create were four fold:

1. to verify the interactive capabilities of Abe and the ability to stream large amounts of data to and from an external source, for control and representation;
2. to demonstrate interactive control of a complex simulation;
3. to use acoustic sound as the control source; and
4. to visually and sonically render the simulation in HD quality image and sound.

The architecture as implemented consisted of the following components: input processing of the violin, converted into control data in Max 5 [12]; transmission through OSC [23] to a standalone client application which relays the data to Abe over a fiber-optic UDP connection; Abe runs a flocking simulation [18] with thousands of entities, sending the simulation data back to the client; the client renders the data visually using the Object-oriented Graphics Rendering Engine (OGRE [14]), simultaneously analyzing the data and sending sonification data back through OSC to Max where it is sonified.

The hardware used for the experiments included: one of two acoustic violins with professional quality internal piezo pickups, a MOTU 828 digital audio interface, a Mac Pro (containing eight 2.8 ghz cores) with a 10-gigabit Myrinet Network Interface Card, three 1080i projectors, a thirty foot, three wall projection screen, a studio two-channel audio system, and Abe. The Mac Pro was connected to Abe over a 10-gigabit network, with tests confirming 6.6-gigabits of bandwidth available for use.

We chose GPL libraries to build the system and original code was written in C++, C with MPI, Java, and Max. OSC provided a rapid and robust method of sending small amounts of data between Max 5 and the graphical client running on the same Mac Pro. We chose the Enet library [6] to transmit data to and from Abe as a low overhead UDP package with controls for data reliability and bandwidth usage. OGRE provides an efficient way of drafting 3D graphical spaces, allowing the compilation of a Mac OS X standalone application to form the client backbone.

3.1. MPI

Development on most supercomputers today is done in FORTRAN, C, or C++ and uses a grid computing API, such as MPI (Message Passing Interface)[13] or charm++ [2]. We chose MPI due to the large body of knowledge and support around its use, as well as its portability to smaller systems (an OpenMPI implementation exists for any multi-core PC and comes bundled with Mac OS X). This allowed the rapid testing of the code on multi-core desktops before delivering it to Abe.

Several significant problems in parallel computing are alleviated by the MPI libraries. When running a large, iterative computational process it is often necessary to copy the relevant data (such as certain control parameters updated in real time) to all the CPUs

involved, recover and merge their results (at least in order to transmit a representation of the internal state back to the user for viewing or audition), and repeat. These operations are referred to as "scattering" and "gathering," respectively. Issues surrounding synchronization of the nodes crop up quickly, such as when one CPU takes much longer to finish than the others, and these are also handled transparently by MPI.

Scattering and gathering can be done in a number of fashions, all appropriate for different tasks. Scattering originates with one process (or "node", with this one typically termed the "root" node), sending data to all nodes (including itself). This can involve the copying of the same data to all nodes, or splitting the data, giving an equal part to each node (as shown in figure 2, below). Gathering does the reverse, taking data from every node (including the root) and sending it to one node where it is ordered into a buffer by the sequence ID of the nodes.

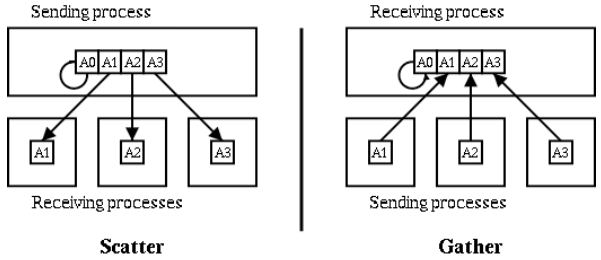

Figure 2. Internode Communication

The root node (A0 in figure 2) sends and receives data from itself as an equal peer, allowing it to take part in the same computation as all the other nodes. In our application we chose to privilege the root node, using it to communicate with controlling clients while the other nodes handle computing the next simulation update (see figure 3).

As per our chosen simulation, see below, we scattered a copy of the full data set to every node at every iteration, but only gathered the portion of the results that were computed by each node. This reduced the amount of data copying required of the root node between iterations, which in turn minimizes the idle time of the compute nodes. More complex partitioning of the computation might have achieved better results, but this will be tackled in future implementations.

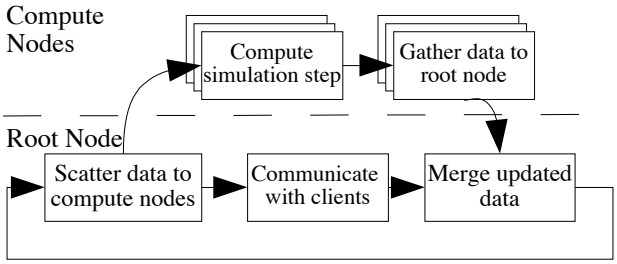

Figure 3. Operation Flow on Abe

3.2. SIMULATION

For the data simulation we chose a flocking algorithm, modeling movement behavior observed in birds and insects [16, 18]. This model exposes a handful of coefficients that enable the production of a wide range of distinctive behaviors. The attraction for our application is both the simplicity of control as well as the extensibility, capable of modeling anywhere from a few to millions of entities. Each entity is stored as six floats and one integer (all 32-bit numbers, for a total of 224 bits or 28 bytes per entity): three for position in 3D space, three for velocity, and the index number of the object. The basic behavior function comprises three primary coefficients: attraction (*a*), alignment (*b*), and crowding (*c*)

$$v = a\,\boldsymbol{x} + b\,\boldsymbol{y} + c\,\boldsymbol{z} + d\,\boldsymbol{w}\ . \tag{1}$$

We add an additional component that attracts entities towards the origin (*w*) proportional to the entity's distance away from the center (*d*), ensuring that all the entities remain in the vicinity of the center of space, rather than drifting away to infinity. The resultant velocity (*v*) is dependent on three computed vectors: the direction (*x*) towards the center of all nearby entities (*p*)

$$x = p - \frac{\sum^{n} p_i}{n}\ , \tag{2}$$

the direction away (*y*) from all nearby entities

$$y = \frac{\sum^{n} (p - p_i)}{n}\ , \tag{3}$$

and the average velocity (*z*) of all nearby entities

$$z = \frac{\sum^{n} v_i}{n}\ . \tag{4}$$

The coefficients (*a* through *d*) are set dynamically for each entity, controlling the impact of each element in determining the behavior of the entity. A neighborhood function determines which entities are "nearby" based on a dynamically updated radius. During each iteration if the number of neighbor entities within the given neighborhood radius is greater than a specified threshold (typically seven) then the neighborhood radius is diminished by 10%. If fewer than the threshold are found the radius is increased by 10% (see pseudo code below). This allows each entity to dynamically acquire its nearest neighbors regardless of the entity density, and discourages all of the entities from clumping into a single cluster. The threshold is stored individually per entity and can be set dynamically, effectively controlling the size of groups that are seen in the simulation.

```
if neighbor_count < threshold then
   neighborhood_radius = neighborhood_radius * 1.1
else if neighbor_count > threshold then
   neighborhood_radius = neighborhood_radius * 0.9
```

Each computation iteration involves the determination of nearest neighbors for all entities, updating their velocities based on the individual coefficients, and the application of the velocity to each entity's position proportional to time

$$p_t = p_{t-1} + v\,\Delta t\ . \tag{5}$$

While all computation nodes require the full data set, broadcast through the scatter step, each is assigned a unique subset of the entities to maintain and update. Each entity maintains its own coefficients and they do

not directly affect other entities, thus they are retained solely on the single compute node the entity is assigned to. Splitting the computation in this way enables a fairly efficient, though not necessarily optimal, parallelization of the simulation. During the gather step each node sends the updated data (positions and velocities) for its assigned entities back to the root node, where they are joined into the full set.

3.2.1. PERFORMANCE

The following figure (fig. 4) depicts the scalability of the algorithm across three hardware configurations. In order to compare across a range of different sized systems we employed the following setups: a MacBook Pro laptop (dual-core Intel 2.2 ghz CPU), a Mac Pro desktop (dual 2.66 ghz six core Intel CPUs), and the supercomputer with varying numbers of CPUs assigned to the simulation.

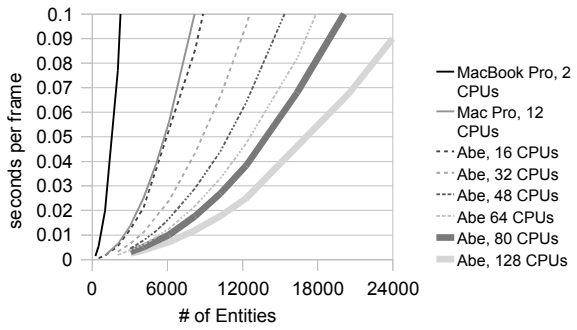

Figure 4. Seconds-per-frame for different hardware configurations.

The target performance speed for any system was fifty updates per second (0.02 seconds-per-frame), to maintain a smooth rendering rate on the display machine. Rates of twenty to twenty-five updates per second (0.05 to 0.04) were considered tolerable, while slower was deemed insufficient. As can be seen in figure 4, the laptop handles ~1000 entities at target speed, the Mac Pro can simulate ~4000, and the super computer runs 10000 and more with a reasonable number of CPUs. While the consumer hardware is capable of running the simulation the HPC machine brings orders of magnitude performance improvements, enabling significantly more complex simulations.

3.3. ACOUSTIC INPUT

The input functionality is implemented as a "zone of control" in the 3D space of the simulation, whereby only entities within the zone are given new coefficients. The center of the zone is initially the origin, but can be moved (using a variety of input devices: keyboard, mouse, or Wii controller). The radius of the zone is set dynamically with each control input.

A control message contains a list of the new simulation coefficients: attraction, alignment, crowding, neighborhood threshold, and the radius of the control zone. These are derived through the linear mapping of acoustic properties in real-time. Using Max 5 the digitized violin sound is analyzed for pitch, amplitude, spectral centroid ("center of mass" of the spectrum, or perceptual "brightness" [17]), and spectral flatness (how tone-like or noise-like the sound is). A number of specific mapping sets were created in order to explore ease of control, all of which consist of simple deterministic connections. The primary mapping used was: pitch to attraction, centroid to alignment, flatness to crowding, pitch and centroid to neighborhood threshold, and amplitude to control zone radius.

The parameters were updated at ~20hz, put into OSC messages and sent to the graphical client. The OSC messages were converted to UDP datagrams and sent directly to the root node of the Abe simulation. The root node then sent the message to all the compute nodes, which checked the zone of control against their assigned entities and changed their coefficients as applicable.

3.4. DISPLAY

The state of the simulation is evident in the changes in position and velocity of the entities, and so a 3D rendering of this data was used as the visual display. The data set was sent out from the root node at every simulation update (~50 times per second) to the graphical client, which then unpacked the data and put it into a format to be rendered. A simple 3D environment was used for the rendering, with a black background, strong downward lighting, and a dozen large spheres placed at significant distance from the origin to give a sense of the virtual space. All of the entities were rendered as initially as white cubes, later replaced with abstract twisted spiral shapes to better depict the direction of facing and movement.

With every update the positions of the entities were updated and the orientations were set to face parallel to the entity's velocity vector (using the shortest rotation from the previous orientation). This provided a quick and efficient visualization of the incoming data from the simulation, and verification of the interactivity.

The data was also sonified, building a sonic environment based on the distributions of the entities in the space. The flocking algorithm employed in the simulation produces strikingly varied clusters and groupings of the entities and thus a sonification algorithm was developed to expose and portray the movements and evolutions within the data.

Employing a modified Adaptive Resonance Theory algorithm [1], the complete data set was traversed every frame, finding groups and creating categories in the simulated space to depict the clustering of the entities. Each entity in question (p) was compared to the set of identified clusters (c, each represented as a float-triplet indicating the center position of the cluster, and where p_x is the position of the control/render view point).

$$\frac{\|c_i - p\|}{\|p_x - p\|} > 1 \qquad (6)$$

If the entity was found to be within a threshold distance of a cluster (formula 6) it was attributed to that cluster and the center of the cluster (c) was moved slightly closer to the new entity (p, formula 7. k controls

the rate of change and incorporation of the new entities, typically $k = 50$).

$$c_t = \frac{(c_{t-1} k + p)}{k+1} \quad (7)$$

If no matches were found in this way a new cluster was created using this entity as the center. The list of identified clusters was preserved between iterations and thus travels with the groups as they move. If no entities are matched to a cluster center after a given update cycle the cluster was deleted from the list.

The list of clusters was then translated into a set of control parameters for voices in a polyphonic digital synthesis engine. The four control parameters were amplitude (mapped to the inverse distance from the camera, p_x), rhythmic speed (controlling an amplitude LFO, mapped to the density of the cluster), a timbrel parameter (controlling the attack of the envelopes, mapped to the *y* axis of the simulated space), and panning (mapped to the *x-z* location in the simulated space relative to the camera, p_x). The pitch of each voice was taken from a pitch history of the acoustic control input, and was retained as long as the particular cluster remained active. When a cluster dispersed and reformed a new pitch was chosen from the history.

In this way coefficient changes were made audible as they quickly transformed the local space, forming clusters with different characteristics, moving, colliding, dispersing, and reforming around the observing controller.

4. DISCUSSION

The final implementation was demonstrated multiple times in the National Center for Supercomputing Applications to display the potential of interactive HPC for musical and artistic applications. While the project required extensive research and development resources it was ultimately operable with only a minimum of setup time (usually thirty minutes). The control of the system, from acoustic sound to simulation to display, was demonstrably apparent. While the actual interaction required an experienced and trained violinist, others were able to verify the functionality by requesting test cases or musical fragments (including the comical entry of popular songs, to witness the simulation effects).

The performance of the simulation, after the debugging and streamlining of the algorithm, was impressive, capable of simulating thousands of entities on a handful of nodes, and up to a million entities on many nodes (see figure 4). However, the system quickly ran out of network bandwidth between the rendering client and Abe as well as rendering power on the client. At 500,000 objects data rates exceeded 5 Gbps, and completely swamped the client, which spent all its time in the network thread and halted graphics updates. In practice the simulation was typically constrained to 50,000 objects or fewer in order to render the graphics quickly (at 50 frames-per-second) on a 1080i projection system.

An unanticipated, and frequently occurring challenge, was discovered with the use of the interactive scheduler running on Abe. The process of starting an interactive job involves queueing a request for a specific number of nodes and amount of time. The request would remain in the queue until the requested processors became available, at which point notification would be received and setup and operation could proceed. Our experience saw wait times anywhere from a minute, for a few nodes and 10 minutes of time, to several hours for larger requests (64 nodes and an hour of time). We were never able to obtain more than 64 nodes, although this was not a hinderance due to the limitations of the renderer. The work around was to progressively queue larger and longer jobs at the beginning of a session, switching between each set of nodes as they became available.

Another side-effect of interactive scheduling is the rotation of server IP addresses. When a job is accepted the requester is given the first nodes that come available, and each node in Abe has a different IP address. Thus configuration scripts had to be edited for each launch, as the returned IP addresses invariably changed every time.

A significant advance for this project and others like it would be to compute the visualization and sonification on the HPC cluster, allowing the consumer computer to act as control relay and display only. This would also enable access from points with smaller network connections (the current implementation requires 1gbps or better). In order to accomplish such remote visualization/sonification many software layers remain to be developed, including distributed, parallel 3D graphical rendering, and HD/CD quality streaming of video and audio out from the supercomputer.

5. ACKNOWLEDGEMENTS

This research was supported in part by the National Science Foundation through TeraGrid resources provided by NCSA under grant number TG-DDM090009.

6. REFERENCES

[1] Carpenter, G. A., S. Grossberg, and D. B. Rosen. Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system. Neural Networks, 4:759–771, 1991.

[2] Charm++. http://charm.cs.uiuc.edu/

[3] Cheng, G., Y. Lu, G. Fox, K. Mills, and T. Haupt. 1993. An interactive remote visualization environment for an electromagnetic scattering simulation on high performance computing system. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, 317-326. Portland, Oregon, United States: ACM.

[4] Cook, Perry. 2001. Principles for designing computer music controllers. In *Proceedings of the 2001 conference on New interfaces for musical expression*, 1-4. Seattle, Washington: National University of Singapore.

[5] David Ellsworth. 2006. Concurrent Visualization in a Production Supercomputing Environment. *IEEE Transactions on Visualization and Computer Graphics* 12: 997-1004.

[6] Enet. http://enet.bespin.org/

[7] Faigle, C., G. C. Fox, W. Furmanski, J. Niemiec, and D.A. Simoni. 1993. Integrating virtual environments with high performance computing. In *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, 62-68.

[8] Hiller, L., and A. Isaacson. 1959. *Experimental Music*. New York: McGraw-Hill.

[9] Höllerer, Tobias, JoAnn Kuchera-Morin, and Xavier Amatriain. 2007. The allosphere: a large-scale immersive surround-view instrument. In *Proceedings of the 2007 workshop on Emerging displays technologies: images and beyond: the future of displays and interacton*, 3. San Diego, California: ACM.

[10] Kaper, Hans G., and Sever Tipei. 1998. Manifold Compositions, Music Visualization, and Scientific Sonification in an Immersive Virtual-Reality Environment. In *International Computer Music Association*, 339–405.

[11] Kriese, C, and S Tipei. 1992. A Compositional approach to additive synthesis on supercomputers. In *Proceedings of the International Computer Music Conference*.

[12] Max 5. http://www.cycling74.com/

[13] MPI. http://www.mcs.anl.gov/research/projects/mpi/

[14] Ogre3D. http://www.ogre3d.org/

[15] Reuther, A., J. Kepner, A. MCcabe, J. Mullen, N.T. Bliss, and Hahn Kim. 2007. Technical Challenges of Supporting Interactive HPC. In *DoD High Performance Computing Modernization Program Users Group Conference, 2007*, 403-409.

[16] Reynolds, C.W. “Flocks, Herds, and Schools: A Distributed Behavioral Model.” Computer Graphics 24:4 (1987) 25–34.

[17] Schubert, Emery; Wolfe, Joe; Tarnopolsky, Alex (2004). Spectral centroid and timbre in complex, multiple instrumental textures. In *Proceedings of the 8th International Conference on Music Perception & Cognition, North Western University, Illinois*. Sydney, Australia: University of New South Wales.

[18] Spector, L., J. Klein, C. Perry, and M. Feinstein. 2003. Emergence of Collective Behavior in Evolving Populations of Flying Agents. In *Proceedings of the Genetic and Evolutionary Computation Conference* (GECCO-2003), pp. 61–73. Berlin: Springer-Verlag.

[19] Tezduyar, T., S. Aliabadi, M. Behr, A. Johnson, V. Kalro, and M. Litke. 1996. Flow simulation and high performance computing. *Computational Mechanics* 18, no. 6 (October 4): 397-412.

[20] Thibodeau, P. n.d. IBM breaks petaflop barrier | Networking - InfoWorld. *Computerworld*. http://www.infoworld.com/t/networking/ibm-breaks-petaflop-barrier-263.

[21] Top500 2010. http://www.top500.org/

[22] Wessel, David, and Matthew Wright. 2002. Problems and Prospects for Intimate Musical Control of Computers. *Computer Music Journal* 26, no. 3: 11-22.

[23] Wright, Matthew. 2005. Open Sound Control: an enabling technology for musical networking. *Organised Sound* 10: 193-200.

[24] Zhao, Zhiming. 2004. An agent based architecture for constructing Interactive Simulation Systems. Ph.D., University of Amsterdam.