GPU Cluster for High Performance Computing

Zhe Fan, Feng Qiu, Arie Kaufman, Suzanne Yoakum-Stover

{fzhe, qfeng, ari, suzi}@cs.sunysb.edu

Center For Visual Computing and Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400

ABSTRACT

Inspired by the attractive Flops/dollar ratio and the incredible growth in the speed of modern graphics processing units (GPUs), we propose to use a cluster of GPUs for high performance scientific computing. As an example application, we have developed a parallel flow simulation using the lattice Boltzmann model (LBM) on a GPU cluster and have simulated the dispersion of airborne contaminants in the Times Square area of New York City. Using 30 GPU nodes, our simulation can compute a 480x400x80 LBM in 0.31 second/step, a speed which is 4.6 times faster than that of our CPU cluster implementation. Besides the LBM, we also discuss other potential applications of the GPU cluster, such as cellular automata, PDE solvers, and FEM.

Keywords: GPU cluster, data intensive computing, lattice Boltzmann model, urban airborne dispersion, computational fluid dynamics

1 INTRODUCTION

The GPU, which refers to the commodity off-the-shelf 3D graphics card, is specifically designed to be extremely fast at processing large graphics data sets (e.g., polygons and pixels) for rendering tasks. Recently, the use of the GPU to accelerate non-graphics computation has drawn much attention [6, 16, 3, 29, 10, 28]. This kind of research is propelled by two essential considerations:

Price/Performance Ratio: The computational power of today's commodity GPUs has exceeded that of PC-based CPUs. For example, the nVIDIA GeForce 6800 Ultra, recently released, has been observed to reach 40 GFlops in fragment processing [11]. In comparison, the theoretical peak performance of the Intel 3GHz Pentium4 using SSE instructions is only 6 GFlops. This high GPU performance results from the following: (1) A current GPU has up to 16 pixel processors and 6 vertex processors that execute 4-dimensional vector floating point instructions in parallel; (2) pipeline constraint is enforced to ensure that data elements stream through the processors without stalls [29]; and (3) unlike the CPU, which has long been recognized to have a memory bottleneck for massive computation [2], the GPU uses fast on-board texture memory which has one order of magnitude higher bandwidth (e.g., 35.2GB/sec on the GeForce 6800 Ultra). At the same time, the booming market for computer games drives high volume sales of graphics cards which keeps prices low compared to other specialty hardware. As a result, the GPU has become a commodity SIMD machine on the desktop that is ready to be exploited for computation exhibiting high compute parallelism and requiring high memory bandwidth.

Evolution Speed: Driven by the game industry, GPU performance has approximately doubled every 6 months since the mid-1990s [15], which is much faster than the growth rate of CPU performance that doubles every 18 months on average (Moore's law), and this trend is expected to continue. This is made possible by the explicit parallelism exposed in the graphics hardware. As the semiconductor fabrication technology advances, GPUs can use additional transistors much more efficiently for computation than CPUs by increasing the number of pipelines.

Recently, the development of GPUs has reached a new high-point with the addition of single-precision 32bit floating point capabilities and the high level language programming interface, called Cg [20]. The developments mentioned above have facilitated the abstraction of the modern GPU as a stream processor. Consequently, mapping scientific computation onto the GPU has turned from initially hardware hacking techniques to more of a high level designing task.

Many kinds of computations can be accelerated on GPUs including sparse linear system solvers, physical simulation, linear algebra operations, partial difference equations, fast Fourier transform, level-set computation, computational geometry problems, and also non-traditional graphics, such as volume rendering, ray-tracing, and flow visualization. (We

SC'04, November 6-12, 2004, Pittsburgh PA, USA
0-7695-2153-3/04 $20.00 (c)2004 IEEE

refer the reader to the web site of General-Purpose Computation Using Graphics Hardware (GPGPU) [1] for more information.) Whereas all of this work has been limited to computing small-scale problems on a single GPU, in this paper we address the large scale computation on a GPU cluster.

Inspired by the attractive Flops/$ ratio and the projected development of the GPU, we believe that a GPU cluster is promising for data-intensive scientific computing and can substantially outperform a CPU cluster at the equivalent cost. Although there have been some efforts to exploit the parallelism of a graphics PC cluster for interactive graphics tasks [9, 13, 14], to the best of our knowledge we are the first to develop a scalable GPU cluster for high performance scientific computing and large-scale simulation. We have built a cluster with 32 computation nodes connected by a 1 Gigabit Ethernet switch. Each node consists of a dual-CPU HP PC with an nVIDIA GeForce FX 5800 Ultra — the GPU that cost $399 in April 2003. By adding 32 GPUs to this cluster, we have increased the theoretical peak performance of the cluster by 512 Gflops at a cost of only $12,768.

As an example application, we have simulated airborne contaminant dispersion in the Times Square area of New York City. To model transport and dispersion, we use the computational fluid dynamics (CFD) model known as the Lattice Boltzmann Method (LBM), which is second-order accurate and can easily accommodate complex-shaped boundaries. Beyond enhancing our understanding of the fluid dynamics processes governing dispersion, this work will support the prediction of airborne contaminant propagation so that emergency responders can more effectively engage their resources in response to urban accidents or attacks. For large scale simulations of this kind, the combined computational speed of the GPU cluster and the linear nature of the LBM model create a powerful tool that can meet the requirements of both speed and accuracy.

In the context of modeling contaminant transport, Brown et al. [4, 5] have presented an approach for computing wind fields and simulating contaminant transport on three different scales: mesoscale, urban scale and building scale. The system they developed, called HIGRAD, computes the flow field by using a second-order accurate finite difference approximation of the Navier-Stokes equations and doing large eddy simulation with a small time step to resolve turbulent eddies. These simulations required a few hours on a supercomputer or cluster to solve a 1.6 km $\times 1.5$ km area in Salt Lake City at a grid spacing of 10 meters (grid resolution: $160 \times 150 \times 36$). In comparison, our method is also second-order accurate, incorporates a more detailed city model, and can simulate the Times Square area in New York City at a grid spacing of 3.8 meters (grid resolution: $480 \times 400 \times 80$) with small vortices in less than 20 minutes.

This paper is organized as follows: Section 2 illustrates how the GPU can be used for non-graphics computing. Section 3 presents our GPU cluster, called the Stony Brook Visual Computing Cluster. In Section 4, we detail our LBM implementation on the GPU cluster, followed by the performance results and a comparison with our CPU cluster. Section 5 presents our dispersion simulation in the Times Square area of New York City. In Section 6, we discuss other potential usage of the GPU cluster for scientific computations. Finally, we conclude in Section 7.

2 GPU Computing Model

A graphics task such as rendering a 3D scene on the GPU involves a sequence of processing stages that run in parallel and in a fixed order, known as the graphics hardware pipeline (see Figure 1). The first stage of the pipeline is the *vertex processing*. The input to this stage is a 3D polygonal mesh. The 3D world coordinates of each vertex of the mesh are transformed to a 2D screen position. Color and texture coordinates associated with each vertex are also evaluated. In the second stage, the transformed vertices are grouped into rendering primitives, such as triangles. Each primitive is scan-converted, generating a set of fragments in screen space. Each fragment stores the state information needed to update a pixel. In the third stage, called the *fragment processing*, the texture coordinates of each fragment are used to fetch colors of the appropriate texels (texture pixels) from one or more textures. Mathematical operations may also be performed to determine the ultimate color for the fragment. Finally, various tests (e.g., depth and alpha) are conducted to determine whether the fragment should be used to update a pixel in the frame buffer.

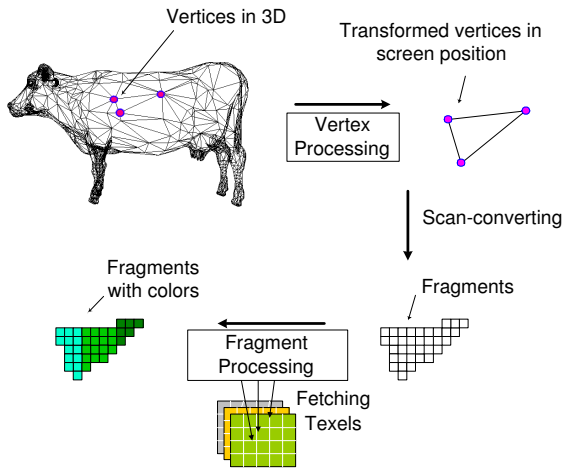

Figure 1: A simplified illustration of the graphics hardware pipeline.

To support extremely fast processing of large graphics data sets (vertices and fragments), modern GPUs (e.g., nVIDIA GeForce and ATI Radeon family cards) employ a stream processing model with parallelism. Currently, up to 6 vertices in the vertex processing stage, and up to 16 fragments in the fragment processing stage can be processed in

parallel by multi-processors. The GPU hardware supports 4-dimensional vectors (representing homogeneous coordinates or the RGBA color channels) and a 4-component vector floating point SIMD instruction set for computation. In addition, the pipeline discipline is enforced that every element in the stream is processed by the similar function and independently of the other elements. This ensures that data elements stream through the pipeline without stalls, and largely account for the high performance gains associated with processing large data sets [29].

Currently, most of the techniques for non-graphics computation on the GPU take advantage of the programmable fragment processing stage. Using the C-like, high-level language, Cg [20], programmers can write fragment programs to implement general-purpose operations. Since fragment programs can fetch texels from arbitrary positions in textures residing in texture memory, a gather operation is supported. Note however, that while the vertex stage is also programmable, it does not support the gather operation. The steps involved in mapping a computation on the GPU are as follows: (1) The data are laid out as texel colors in textures; (2) Each computation step is implemented with a user-defined fragment program which can include gather and mathematic operations. The results are encoded as pixel colors and rendered into a pixel-buffer (a buffer in GPU memory which is similar to a frame-buffer); (3) Results that are to be used in subsequent calculations are copied to textures for temporary storage.

For general-purpose computation on the GPU, an essential requirement is that the data structure can be arranged in arrays in order to be stored in a 2D texture or a stack of 2D textures. For a matrix or a structured grid, this layout in texture is natural. Accommodating more complicated data structures may require the use of *indirection textures* that store texture coordinates used to fetch texels from other textures. For example, to store a static 2D binary tree, all the nodes can be packed into a 2D texture in row-priority order according to the node IDs. Using two indirection textures, the texture coordinates of each node's left child and right child can be stored. However, lacking pointers in GPU programs makes computations that use some other complex data structures (i.e., dynamic link list) difficult for the GPU. GPU computation may also be inefficient in cases where the program control flow is complex. It is also the case that the GPU on-board texture memory is relatively small (currently the maximum size is 256MB). In our previous work with LBM simulation on a single GeForce FX 5800 Ultra with 128MB texture memory, we found that at most 86MB texture memory can actually be used to store the computational lattice data. As a result, our maximum lattice size was 92^3. Fortunately, many massive computations exhibit the feature that they only require simple data structures and simple program control flows. By using a cluster of GPUs, these computations can reap the benefits of GPU computing while avoiding its limitations.

3 Our GPU Cluster

The Stony Brook Visual Computing Cluster (Figure 2) is our GPU cluster built for two main purposes: as a GPU cluster for graphics and computation and as a visualization cluster for rendering large volume data sets. It has 32 nodes connected by a 1 Gigabit Ethernet switch (Actually, the cluster has 35 nodes, but only 32 are used in this project). Each node is an HP PC equipped with two Pentium Xeon 2.4GHz processors and 2.5GB memory. Each node has a GPU, the GeForce FX 5800 Ultra with 128MB memory, used for GPU cluster computation. Each node also has a volume rendering hardware (VolumePro 1000) and currently 9 of the nodes have also HP Sepia-2A composting cards with fast ServerNet [25] for rendering large volume data sets. Each node can boot under Windows XP or Linux, although our current application of the GPU cluster runs on Windows XP.

Figure 2: The Stony Brook Visual Computing Cluster.

The architecture of our GPU cluster is shown as Figure 3. We use MPI for data transfer across the network during execution. Each port of the switch has 1 Gigabit bandwidth. Besides network transfer, data transfer includes upstreaming data from GPU to PC memory and downstreaming data from PC memory to GPU for the next computation. This communication occurs over an AGP 8x bus, which has been well known to have an asymmetric bandwidth (2.1GB/sec peak for downstream and 133MB/sec peak for upstream). The asymmetric bandwidth reflects the need for the GPU to push vast quantities of graphics data at high speed and to read back only a small portion of data. As shown in Section 4.4, the slower upstream transfer rate slows down the entire communication. Recent exciting news indicates that this situation will be improved with the PCI-Express bus to be available later this year [30]. By connecting with a x16 PCI-Express slot, a graphics card can communicate with the system at 4GB/sec in both upstream and downstream directions. Moreover, the PCI-Express will allow multiple GPUs to be plugged into one PC. The interconnection of these GPUs will greatly reduce the network load.

Currently, we only use the fragment processing stage of the GeForce FX 5800 Ultra for computing, which features

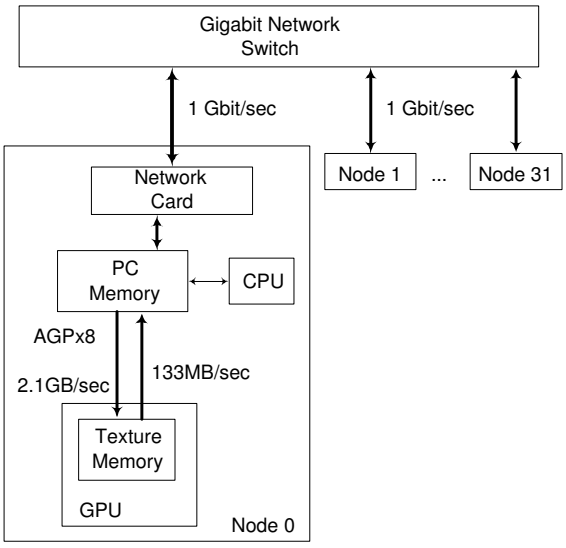

Figure 3: The architecture of our GPU cluster. (Although all 32 nodes have the same configuration, we show only node 0 in detail.)

a theoretical peak of 16 Gflops, while the dual-processor Pentium Xeon 2.4GHz reaches approximately 10 Gflops. The theoretical peak performance of our GPU cluster is $(16 + 10) \times 32 = 832$ Gflops. Although the whole GPU cluster cost was about $136,000 (excluding the VolumePro cards and the Sepia cards which are not used here), this price can be decreased by designing the system specifically for the purpose of GPU cluster computation, since the large memory configurations and the dual processors of the PCs in this cluster actually do not improve the performance of GPU computing. Stated in another way, by plugging 32 GPUs into this cluster, we increase its theoretical peak performance by $16 \times 32 = 512$ GFlops at a price of $\$399 \times 32 = \$12,768$. We therefore get in principle 41.1 Mflops peak/$.

4 Parallel LBM Computation on the GPU Cluster

In this section we describe the first example application, parallel LBM computation that we developed on the GPU cluster. We begin this section with a brief introduction to the LBM model and then review our previous work of mapping the computation onto a single GPU. Afterwards, we present the algorithm and network optimization techniques for scaling the model onto our GPU cluster and report the performance in comparison with the same model executed on the CPU cluster.

4.1 LBM Flow Model

The LBM is a relatively new approach in computational fluid dynamics for modeling gases and fluids [26]. Developed principally by the physics community, the LBM has been applied to problems of flow and reactive transport in porous media, environmental science, national security, and others. The numerical method is highly parallelizable, and most notably, it affords great flexibility in specifying boundary shapes. Even moving and time-dependent boundaries can be accommodated with relative ease [24].

The LBM models Boltzmann dynamics of "flow particles" on a regular lattice. Figure 4 shows a unit cell of the D3Q19 lattice, which includes 19 velocity vectors in three-dimension (the zero velocity in the center site and the 18 velocities represented by the 6 nearest axial and 12 second-nearest minor diagonal neighbor links). Associated with each lattice site, and corresponding to each of the 19 velocities are 19 floating point variables, f_i, representing velocity distributions. Each distribution represents the probability of the presence of a fluid particle with the associated velocity.

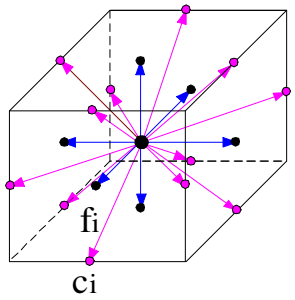

Figure 4: The D3Q19 LBM lattice geometry. The velocity distribution f_i is associated with the link vector $\mathbf{c}_i$.

The Boltzmann equation expresses how the average number of flow particles move between neighboring sites due to inter-particle interactions and ballistic motion. This dynamics can be represented as a two-step process of *collision* and *streaming*. Particles stream synchronously along links in discrete time steps. Between streaming steps, the Bhatnager, Gross, Krook (BGK) model is used to model collisions as a statistical redistribution of momentum, which locally drives the system toward equilibrium while conserving mass and momentum [31]. Complex shaped boundaries such as curves and porous media can be represented by the location of the intersection of the boundary surfaces with the lattice links [24]. The LBM is second-order accurate in both time and space, with an advection-limited time step. In the limit of zero time step and lattice spacing, LBM yields the Navier-Stokes equation for an incompressible fluid.

The LBM model can be further extended to capture thermal effects as in convective flows. A hybrid thermal model has been recently developed [17]. The hybrid thermal LBM (HTLBM) abandons the BGK collision model for the more stable Multiple Relaxation Time (MRT) collision model [8]. Temperature, modeled with a standard diffusion-advection equation implemented as a finite difference equation is coupled to the MRT LBM via an energy term. Ultimately, the

implementation of the HTLBM is similar to the earlier LBM requiring only two additional matrix multiplications.

4.2 LBM on a Single GPU

In a previous work [18], our group have implemented a BGK LBM simulation on the nVIDIA GeForce4 GPU, which has a non-programmable fragment processor, using complex texture operations. Since then we have ported the BGK LBM computation to newer graphics hardware, the GeForce FX, and have achieved about 8 times faster speed on the GeForce FX 5900 Ultra compared to the software version running on Pentium IV 2.53GHz without using SSE instructions. The programmability of the GeForce FX makes porting to the GPU straightforward and efficient. Because our latest parallel version on the GPU cluster is based on it, we briefly review the single GPU implementation on the GeForce FX.

As shown in Figure 5, to lay out the LBM data, the lattice sites are divided into several volumes. Each volume contains data associated with a given state variable and has the same resolution as the LBM lattice. For example, each of the 19 velocity distributions f_i in D3Q19 LBM, is represented in a volume. To use the GPU vector operations and save storage space, we pack four volumes into one stack of 2D textures (note that a fragment or a texel has 4 color components). Thus, the 19 distribution values are packed into 5 stacks of textures. Flow densities and flow velocities at the lattice sites are packed into one stack of textures in a similar fashion.

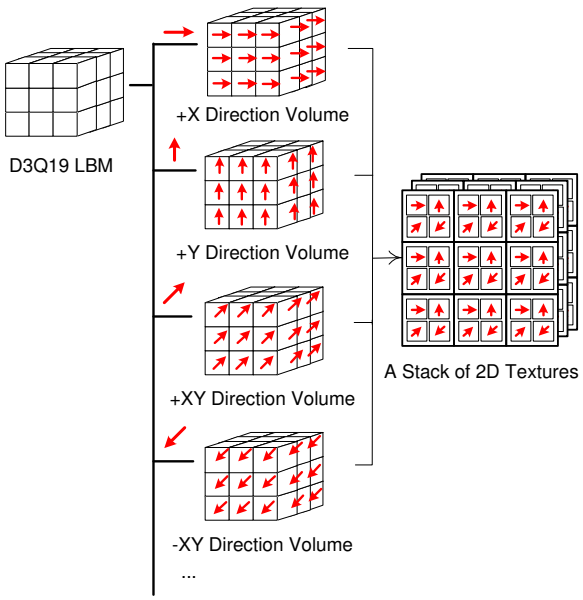

Figure 5: Each velocity distribution f_i, associated with a given direction, is grouped into a volume. We pack every four volumes into one stack of 2D textures.

Boundary link information (e.g., flags indicating whether the lattice links intersect with boundary surfaces along with the intersection positions) is also stored in textures. However, since most links do not intersect the boundary surface, we do not store boundary information for the whole lattice. Instead, we cover the boundary regions of each Z slice using multiple small rectangles. Thus, we need to store the boundary information only inside those rectangles in 2D textures.

The LBM operations (e.g., streaming, collision, and boundary conditions) are translated into fragment programs to be executed in the rendering passes. For each fragment in a given pass, the fragment program fetches any required current lattice state information from the appropriate textures, computes the LBM equations to evaluate the new lattice states, and renders the results to a pixel buffer. When the pass is completed, the results are copied back to textures for use in the next step.

4.3 Scaling LBM onto the GPU Cluster

To scale LBM onto the GPU cluster, we choose to decompose the LBM lattice space into sub-domains, each of which is a 3D block. As shown in Figure 6, each GPU node computes one sub-domain. In every computation step, velocity distributions at border sites of the sub-domain may need to stream to adjacent nodes. This kind of streaming involves three steps: (1) Distributions are read out from the GPU; (2) They are transferred through the network to appropriate neighboring nodes; (3) They are then written to the GPU in the neighboring nodes. For ease of discussion, we divide these across-network streaming operations into two categories: streaming axially to nearest neighbors (represented by black arrows in Figure 6) and streaming diagonally to second-nearest neighbors (represented by blue arrows). Note that although Figure 6 only demonstrates 9 sub-domains arranged in 2 dimensions, our implementation is scalable and functions in a similar fashion for sub-domains arranged in 3 dimensions.

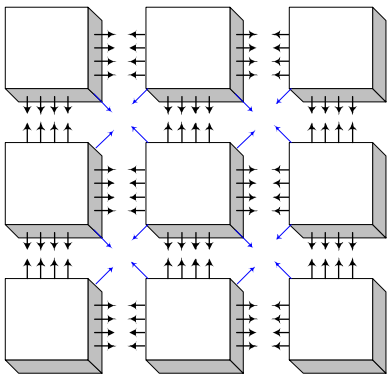

Figure 6: Each block represents a sub-domain of the LBM lattice processed by one GPU. Velocity distributions at border sites stream to adjacent nodes at every computation step. Black arrows indicate velocity distributions that stream axially to nearest neighbor nodes while blue arrows indicate velocity distributions that stream diagonally to second-nearest neighbor nodes.

The primary challenge in scaling LBM computation onto the GPU cluster is to minimize the communication cost — the time taken for network communication and for transferring data between the GPU and the PC memory. Overlapping network communication time with the computation time is feasible, since the CPU and the network card are all standing idle while the GPU is computing. However, because each GPU can compute its sub-domain quickly, optimizing network performance to keep communication time from becoming the bottleneck is still necessary. Intuitively one might want to minimize the size of transferred data. One way to do this is to make the shape of each sub-domain as close as possible to a cube, since for block shapes the cube has the smallest ratio between boundary surface area and volume. Another idea that we have not yet studied is to employ lossless compression of transferred data by exploiting space coherence or data coherence between computation steps. We have found, however, that other issues actually dominate the communication performance.

The communication switching time has a significant impact on network performance. We performed experiments on the GPU cluster using MPI and replicated these experiments using communication code that we developed using TCP/IP sockets. The results were the same: (1) During the time when a node is sending data to another node, if a third node tries to send data to either of those nodes, the interruption will break the smooth data transfer and may dramatically reduce the performance; (2) Assuming the total communication data size is the same, a simulation in which each node transfers data to more neighbors has a considerably larger communication time than a simulation in which each node transfers to fewer neighbors.

To address these issues, we have designed communication schedules [27] that reduce the likelihood of interruptions. We have also further simplified the communication pattern of the parallel LBM simulation. In our design, the communication is scheduled in multiple steps and in each step certain pairs of nodes exchange data. This schedule and pattern are illustrated in Figure 7 for 16 nodes arranged in 2 dimensions. The same procedure works for configurations with more nodes and for 3D arrangement as well. The different colors represent the different steps. In the first step, all nodes in the $(2i)th$ columns exchange data with their neighbors to the left. In the second step, these nodes exchange data with neighbors to the right. In the third and fourth steps, nodes in the $(2i)th$ rows exchange data with their neighbors above and below, respectively. Note that LBM computation requires that nodes need to exchange data with their second-nearest neighbors too. There are as many as 4 second-nearest neighbors in 2D arrangements and up to 12 in 3D D3Q19 arrangements. To keep the communication pattern from becoming too complicated, and to avoid additional overhead associated with more steps, we do not allow direct data exchange diagonally between second-nearest neighbors. Instead, we transfer those data indirectly in a two-step process. For example, as shown in Figure 7, data that node B wants to send to node E will first be sent to node A in step 1, then be sent by node A to node E in step 3. If the sub-domain in a GPU node is a lattice of size N^3, the size of the data that it sends to a nearest neighbor is $5N^2$, while the data it sends to a second-nearest neighbor has size of only N. Using the indirect pattern increases the packet size between nearest neighbors only by $\frac{c}{5N}$ (c is 1 or 2 for 2D arrangement and 1-4 for 3D arrangement). Since the communication pattern is also greatly simplified, particularly for 3D node arrangements, the network performance is greatly improved.

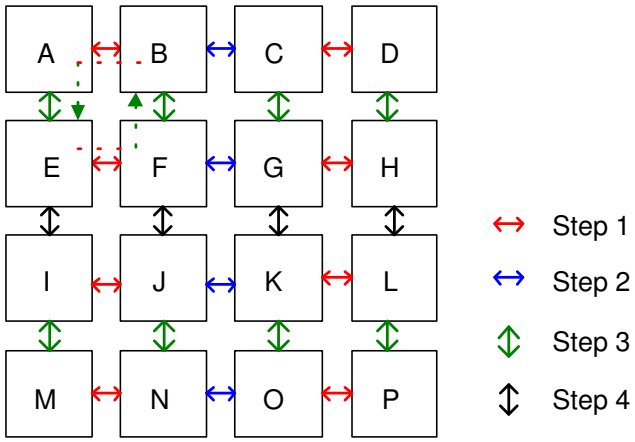

Figure 7: The communication schedule and pattern of parallel LBM Simulation. Different colors indicate the different steps in the schedule.

We also found that for simulations with a small number of nodes (less than 16), synchronizing the nodes by calling MPI_barrier() at each scheduled step improves the network performance. However, if more than 16 nodes are used, the overhead of the synchronization overwhelms the performance gained from the synchronized schedule.

The data transfer speed from GPU to CPU represents another bandwidth limitation. Because of the way that we map the data to textures (described in Section 4.2), the velocity distributions that stream out of the sub-domain are stored in different texels and different channels in multiple textures. We have designed fragment programs which run in every time step to gather together into a texture all these data. Then they are read from the GPU in a single read operation (e.g., OpenGL function glGetTexImage()). In so doing, we minimize the overhead of initializing the read operations. As described in Section 3, this bandwidth limitation will be ameliorated later this year when the PCI-Express bus becomes available on the PC platform.

4.4 Performance of LBM on the GPU Cluster

In addition to the GPU cluster implementation, we have implemented the parallel LBM on the same cluster using

Table 1: Per step execution time (in ms) for CPU and GPU clusters and the GPU cluster / CPU cluster speedup factor. Each node computes an 80^3 sub-domain of the lattice.

Number of nodes	CPU cluster	GPU cluster				Speedup
	Total	Computation	GPU and CPU Communication	Network Communication: Non-overlapping Cost (Total)	Total	
1	1420	214	-	-	214	6.64
2	1424	216	13	0 (38)	229	6.22
4	1430	224	42	0 (47)	266	5.38
8	1429	222	50	0 (68)	272	5.25
12	1431	230	50	0 (80)	280	5.11
16	1433	235	50	0 (85)	285	5.03
20	1436	237	50	0 (87)	287	5.00
24	1437	238	50	0 (90)	288	4.99
28	1439	237	50	11 (131)	298	4.83
30	1440	237	50	25 (145)	312	4.62
32	1440	237	49	31 (151)	317	4.54

the CPUs. The time and work taken to develop and optimize these two implementations were similar (about 3 man-months each). Note that although each node has two CPUs, for the purpose of a fair comparison, we used only one thread (hence one CPU) per node for computation.

In Table 1, we report the simulation execution time per step (averaged over 500 steps) in milliseconds on both the CPU cluster and the GPU cluster with 1, 2, 4, 8, 16, 20, 24, 28, 30 and 32 nodes. Each node evaluates an 80^3 sub-domain and the sub-domains are arranged in 2 dimensions. The timing for the CPU cluster simulation (shown in column 2 of table 1) includes only computation time because the network communication time was overlapped with the computation by using a second thread for network communication. The timing for the GPU cluster simulation (shown in column 6) includes: computation time, GPU and CPU communication time, and non-overlapping network communication time. Note that the computation time also includes the time for boundary condition evaluation for the city model described in Section 5. As the boundary condition evaluation time is only a small portion of the computation time, the computation time is similar for all the nodes. Network communication time (plotted as a function of the number of nodes in Figure 8) was partially overlapped with the computation because we let each GPU compute collision operation on inner cells of its sub-domain (which takes roughly 120 ms) simultaneously with network communication. If the network communication time exceeds 120 ms, the remainder will be non-overlapping and affect the simulation time. In column 5 we show this remainder cost along with a total network communication time in parenthesis.

The GPU cluster / CPU cluster speedup factor is plotted as a function of the number of nodes in Figure 9. When

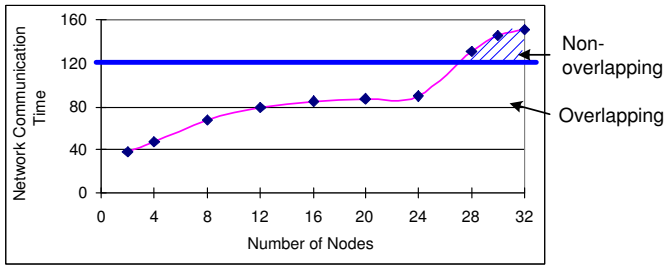

Figure 8: The network communication time measured in ms. The area under the blue line represents the part of network communication time which was overlapped with computation. The shadow area represents the remainder.

only a single node is used, the speedup factor is 6.64. This value projects the theoretical maximum GPU cluster / CPU cluster speedup factor which could be reached if all communication bottlenecks were eliminated by better optimized network and larger GPU/CPU bandwidth. When the number of nodes is below 28, the network communication will be totally overlapped with the computation. Accordingly the growth of the number of nodes only marginally increases the execution time due to the GPU/CPU communication and the curve flattens approximately at 5. When the number of nodes increase to 28 or above, the network can't be totally overlapped, resulting in a drop in the curve.

Three enhancements can further improve this speedup factor without changing the way that we map the LBM computation onto the GPU cluster: (1) Using a faster network, such as Myrinet. (2) Using the PCI-Express bus that will be available later this year to achieve faster communication between the GPU and the system and to plug multiple GPUs into each PC. (3) Using GPUs with larger texture memories

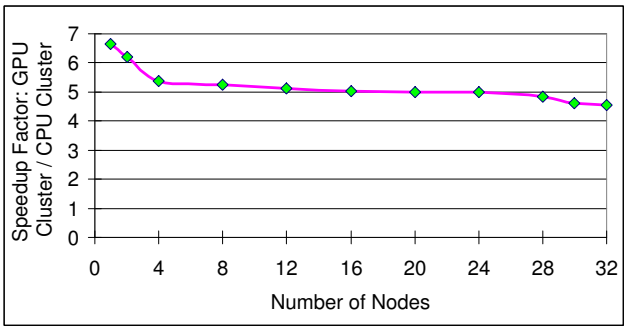

Figure 9: Speedup factor of the GPU cluster compared with the CPU cluster.

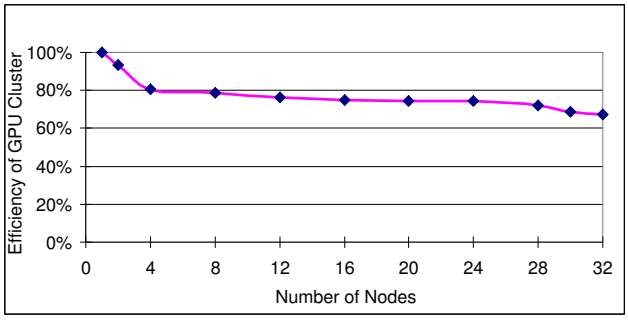

Figure 10: Efficiency of the GPU cluster with respect to the number of nodes.

(currently, larger memories of 256MB are available) so that each GPU can compute a larger sub-domain of the lattice and thereby increase the computation/communication ratio. Further GPU development, and the consequent increase in performance, will serve to improve the speedup factor even further (Note that today's GeForce 6800 Ultra, which has been observed to reach 40 GFlops in fragment processing, is already at least 2.5 times faster than the GeForce FX 5800 Ultra in our cluster). On the other hand, our CPU cluster implementation could be further optimized too by using SSE instructions, which we are going to implement in the near future. With this optimization, the CPU cluster computation is supposed to be about 2 to 3 times faster.

To quantify the scalability of the GPU cluster, Table 2 shows the computed efficiency of the GPU cluster as a function of the number of nodes. The efficiency values are also plotted in Figure 10.

Table 2: The GPU cluster computational power and the efficiency with respect to the number of nodes.

Number of Nodes	Number of cells computed per second	Speedup	Efficiency
1	2.3M	–	–
2	4.3M	1.87	93.5%
4	7.3M	3.17	79.3%
8	14.4M	6.26	78.3%
12	20.9M	9.09	75.8%
16	27.4M	11.91	74.4%
20	34.0M	14.78	73.9%
24	40.7M	17.70	73.8%
28	45.9M	19.96	71.3%
30	47.0M	20.43	68.1%
32	49.2M	21.39	66.8%

Our simulation computes $640 \times 320 \times 80 = 15.6M$ LBM cells in 0.317 second/step using 32 GPU nodes, resulting in 49.2M cells/second. This performance is comparable with supercomputers [21, 22, 23]. In the work of Martys et al. [21], $128 \times 128 \times 256 = 4M$ LBM cells were computed in about 5 seconds/step on IBM SP2 using 16 processors, which corresponds to 0.8M cells/second. In 2002, Massaioli and Amati [22] reported the optimized D3Q19 BGK LBM running on 16 IBM SP Nodes (16-way Nighthawk II nodes, Power3@375MHz) with 16GB shared memory using OpenMP. They computed $128 \times 128 \times 256 = 4M$ LBM cells in 0.26 second/step, which is 15.4M cells/second. They were able to further increase this performance to 20.0M cells/second using more sophisticated optimization techniques, such as (1) "fuse" the streaming and collision steps to reduce the memory accesses; (2) keep distributions "at rest" in memory and implement the streaming by the indexes translation; (3) bundle the distributions in a way that relieves the Segment Lookaside Buffer (SLB) and Translation Lookaside Buffer (TLB) activities during address translation. In 2004, by using the above sophisticated optimization techniques and further taking advantage of vector codes, they achieved the performance of 108.1M cells/second on 32 processors with Power4 IBM [23]. Still, the GPU cluster is competitive with supercomputers at a substantially lower price.

In the above discussion, we have chosen to fix the size of each sub-domain as to maximize the performance of each GPU node. This means, using more nodes we can obtain more cycles to compute larger lattices within a similar time frame. However, another performance criterion for a cluster is to keep the problem size fixed, but increase the number of nodes to achieve a faster speed. However, we have found that in doing so, the sub-domains become smaller, resulting in a low computation/communication ratio. As a consequence, the network performance becomes the bottleneck. We thus may need a faster network in order to better exploit the computational power of the GPUs. We have tested this performance criterion with a $160 \times 160 \times 80$ lattice and started with 4 nodes. When the number of nodes increases from 4 to 16, the GPU cluster / CPU cluster speedup factor drops from 5.3 to 2.4. When more nodes are used, the GPU cluster and the CPU cluster gradually converge to achieve comparable performance.

5 DISPERSION SIMULATION IN NEW YORK CITY

Using the LBM, we have simulated on our GPU cluster the transport of airborne contaminants in the Times Square area of New York City. As shown in Figure 11, this area extends North from 38th Street to 59th Street, and East from the 8th Avenue to Park Avenue.

Figure 11: The simulation area shown on the Manhattan map, enclosed by the blue contour. This area extends North from 38th Street to 59th Street, and East from the 8th Avenue to Park Avenue. It covers an area of about 1.66 km $\times$ 1.13 km, consisting of 91 blocks and roughly 850 buildings.

The geometric model of the Times Square area that we use is a 3D polygonal mesh that has considerable details and accuracy (see Figure 12). It covers an area of about 1.66 km $\times 1.13$ km, consisting of 91 blocks and roughly 850 buildings. We model the flow using the D3Q19 BGK LBM with a $480 \times 400 \times 80$ lattice. This simulation is executed on 30 nodes of the GPU cluster (each node computes an 80^3 subdomain). The urban model is rotated to align it with the LBM domain axes. It occupies a lattice area of 440×300 on the ground. As a result, the simulation resolution is about 3.8 meters / lattice spacing. We simulate a northeasterly wind with a velocity boundary condition on the right side of the LBM domain. The LBM flow model runs at 0.31 second/step on the GPU cluster. After 1000 steps of LBM computation, the pollution tracer particles begin to propagate along the LBM lattice links according to transition probabilities obtained from the LBM velocity distributions [19].

Figure 12 shows the velocity field visualized with streamlines at time step 1000. The blue color streamlines indicates that the direction of velocity is approximately horizontal, while the white color indicates a vertical component in the velocity as the flow passes over the buildings. Figure 13 shows the dispersion simulation snapshot with volume rendering of the contaminant density.

Currently, we render the images off-line. In the future, we plan to make better use of the GPUs by rendering the results on-line. A potential advantage of the GPU cluster is that the on-line visualization is feasible and efficient. Since the simulation results already reside in the GPUs, each node could rapidly render its contents, and the images could then be transferred through a specially designed composing network to form the final image. HP is already developing new technology [12] for its Sepia PCI cards [25], that can read out data from the GPU through the DVI port and transfer them at a rate of 450-500 MB/second in its composing network. This feature will enable immediate visual feedback for computational steering.

6 DISCUSSION: OTHER COMPUTATIONS ON THE GPU CLUSTER

As discussed in Section 1, many kinds of computations have been ported to the GPU. Many of these have the potential to run on a GPU cluster as well. The limitations lie in the inability to efficiently handle complex data structures and complex control flows. One approach to this problem is to let the GPU and CPU work together, each doing the job that it does best. This has been illustrated by Carr et al. [7], who used the CPU to organize the data structure and the GPU to compute ray-triangle intersections. This hybrid computation makes it possible to apply the GPU cluster to more computational problems. Since our main focus is flow simulation, in the following we discuss the possibility of computing cellular automata, explicit and implicit PDE methods, and FEM on the GPU cluster.

Since the LBM is a kind of explicit numerical method on a structured grid, we expect that the GPU cluster computing can be applied to the entire class of explicit methods on structured grids and cellular automata as well. For explicit methods on unstructured grids, the main challenge is to represent the grid in textures. If the grid connection does not change during computation, the structure can be laid out in textures in a preprocessing step. The data associated with the grid points can be laid out in textures in the order of point IDs. Using indirection textures, the texture coordinates of neighbors of each point can also be stored. Hence, accessing neighbor variables will require two texture fetch operations. The first operation fetches the texture coordinates of the neighbor. Using the coordinates, the second operation fetches the neighbor variables.

To parallelize explicit methods on the GPU cluster, the domain can be decomposed into local sub-domains (see Figure 14). For each GPU node, we denote the grid points inside its sub-domain as *local points* and the grid points outside its sub-domain but whose variables are needed to be accessed as *neighbor points*. All other points are called *external points*. Non-local gather operations, which involve accessing the data of neighbor points, can be achieved as a local gather operation by adding *proxy points* at the computation boundary to store the variables of neighbor points obtained over the network.

Implicit finite differences and FEM require the solution of a large sparse linear system, $Ax = y$. Krüger and Westermann [16] and Boltz et al. [3] have implemented iterative methods for solving sparse linear systems such as conjugate gradient and Gauss-Seidel on the GPU. To scale their ap-

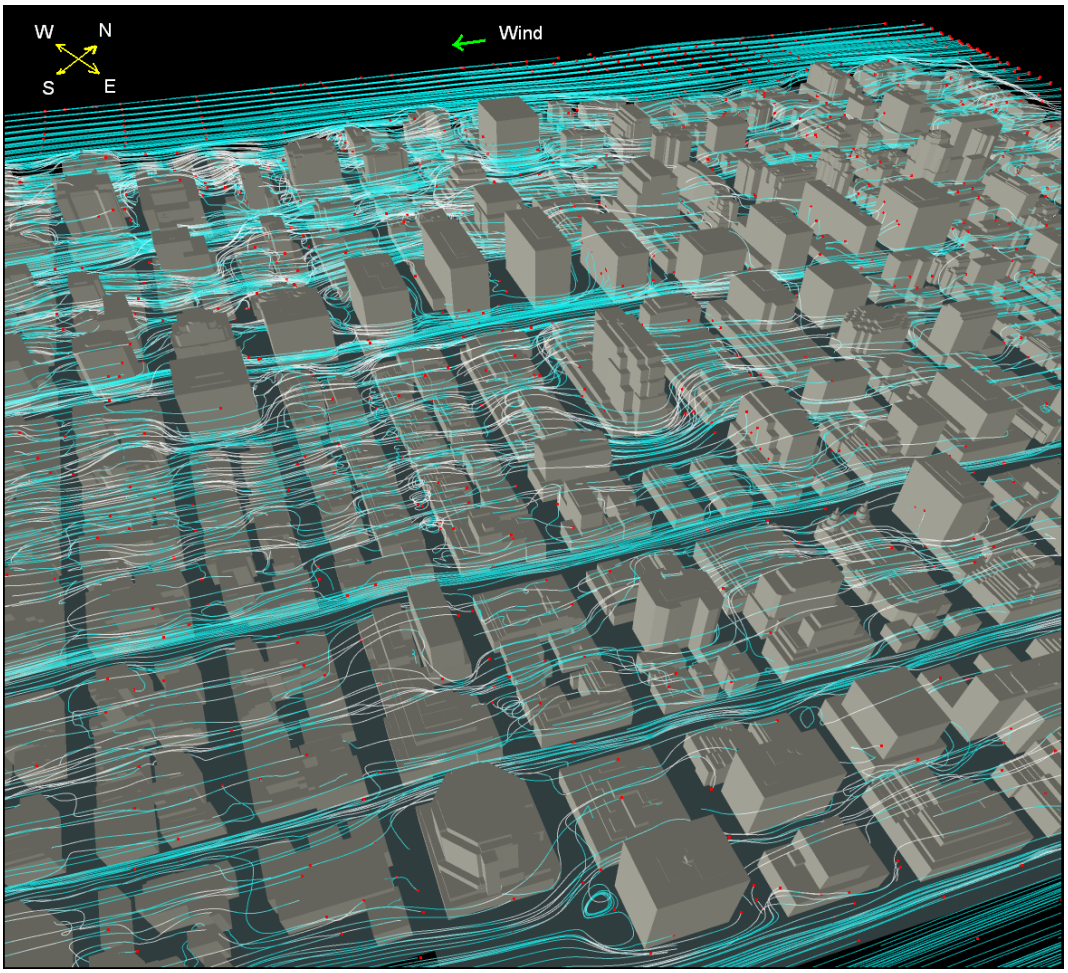

Figure 12: A snapshot of the simulation of air flow in the Times Square area of New York City at time step 1000, visualized by streamlines. The blue color indicates that the direction of velocity is approximately horizontal, while the white color indicates a vertical component in the velocity as the flow passes over buildings. Red points indicate streamline origins. Simulation lattice size is $480 \times 400 \times 80$. (Only a portion of the simulation volume is shown in this image.)

proach to the GPU cluster, in addition to decomposing the domain, the matrix and vector need to be decomposed so that matrix vector multiplies can be executed in parallel. In the case of a sparse linear system, the matrix and vector may be decomposed using an approach similar to one developed for a CPU cluster [32]. In each cluster node, the local matrix includes those matrix rows which correspond to local points, and the local vector includes those vector elements which correspond to the local and neighbor (proxy) points (see Figure 15). In each iteration step, the network communication is needed to read the vector elements corresponding to neighbor points in order to update proxy point elements in the local vector. Then, the local matrix and local vector multiple is executed and the result is the vector corresponding to local points. Since each time-step takes several iteration steps, although the network communication to local computation ratio is still at the order of $O(\frac{1}{N})$, the actual value of this ratio may be larger than for explicit methods on the GPU cluster.

7 Conclusions

In this paper, we propose the use of a cluster of commodity GPUs for high performance scientific computing. Adding 32 GPUs to a CPU cluster for computation increases the theoretical peak performance by 512 Gflops at the cost of $12,768. To demonstrate the GPU cluster performance, we used the LBM to simulate the transport of airborne contaminants in the Times Square area of New York City with a resolution of 3.8 meters and performance of 0.31 second/step on 30 nodes. Compared to the same model implemented on the CPU cluster, the speed-up is above 4.6 and better performance is anticipated. Considering the rapid evolution of GPUs, we believe that the GPU cluster is a very promising machine for scientific computation. Our approach is not limited to LBM, and we also discussed methods for implementing other numerical methods on the GPU cluster including cellular automata, finite differences, and FEM.

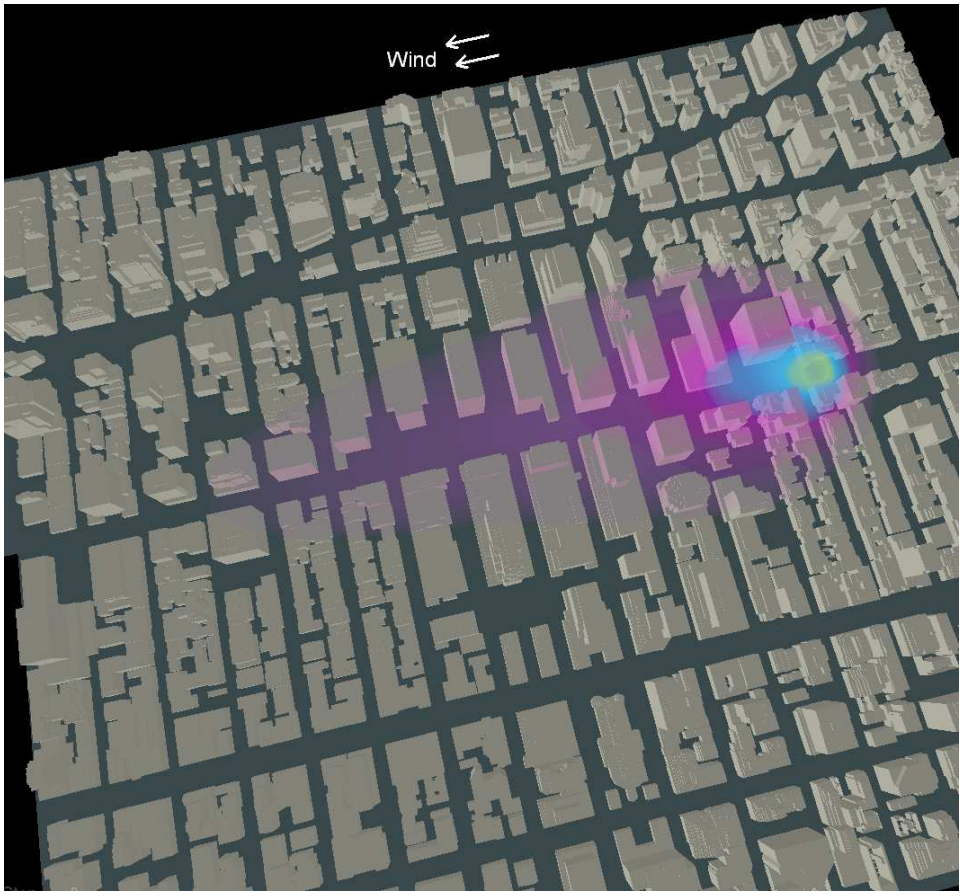

Figure 13: A snapshot of the simulation of air flow in the Times Square area with dispersion density volume rendered.

8 ACKNOWLEDGEMENTS

This work has been supported by an NSF grant CCR-0306438 and a grant from the Department of Homeland Security, Environment Measurement Lab. We would like to thank Bin Zhang for setting up and maintaining the Stony Brook Visual Computing Cluster. We also thank Li Wei for his early work on the single GPU accelerated LBM, and Ye Zhao, Xiaoming Wei and Klaus Mueller for helpful discussions on LBM related issues. Finally, we would like to acknowledge HP and Terarecon for their contributions and help with our cluster.

REFERENCES

[1] General-Purpose Computation Using Graphics Hardware (GPGPU). http://www.gpgpu.org.

[2] J. Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *ACM Turing Award Lecture*, 1977.

[3] J. Bolz, I. Farmer, E. Grinspun, and P. Schröder. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph. (SIGGRAPH)*, 22(3):917–924, 2003.

[4] M. Brown, M. Leach, R. Calhoun, W.S. Smith, D. Stevens, J. Reisner, R. Lee, N.-H. Chin, and D. DeCroix. Multiscale modeling of air flow in Salt Lake City and the surrounding region. *ASCE Structures Congress*, 2001. LA-UR-01-509.

[5] M. Brown, M. Leach, J. Reisner, D. Stevens, S. Smith, H.-N. Chin, S. Chan, and B. Lee. Numerical modeling from mesoscale to urban scale to building scale. *AMS 3rd Urb. Env. Symp.*, 2000.

[6] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for GPUs: Stream Computing on Graphics Hardware. *ACM Trans. Graph. (SIGGRAPH), to appear*, 2004.

[7] N. A. Carr, J. D. Hall, and J. C. Hart. The ray engine. *Proceedings of Graphics Hardware*, pages 37–46, September 2002.

[8] D. D'Humieres, M. Bouzidi, and P. Lallemand. Thirteen-velocity three-dimensional lattice Boltzmann model. *Phys. Rev. E*, 63(066702), 2001.

[9] N. K. Govindaraju, A. Sud, S.-E. Yoon, and D. Manocha. Interactive visibility culling in complex environments using occlusion-switches. In *Proceedings Symposium on Interactive 3D Graphics*, pages 103–112, 2003.

[10] M. Harris, G. Coombe, T. Scheuermann, and A. Lastra. Physically-based visual simulation on graphics hardware. *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 109–118, September 2002.

[11] M. J. Harris. GPGPU: Beyond graphics. *Eurographics Tutorial*, August 2004.

[12] A. Heirich, P. Ezolt, M. Shand, E. Oertli, and G. Lupton. Per-

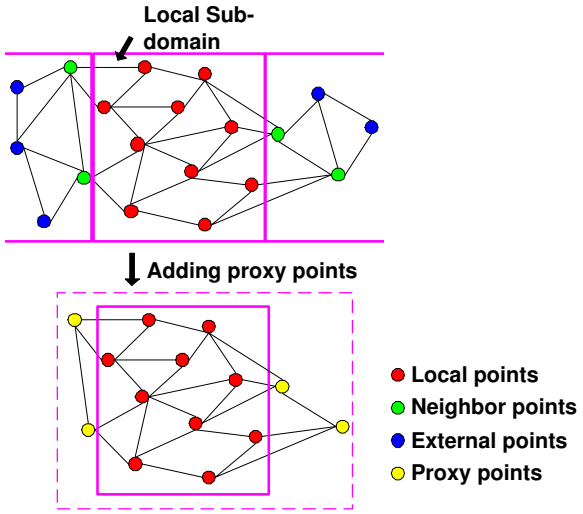

Figure 14: Decomposing the grid and adding proxy points to support non-local gather operations

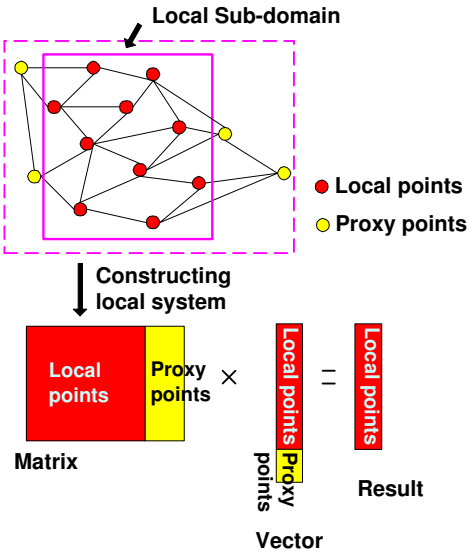

Figure 15: Decomposition of a matrix and a vector to implement matrix vector multiplies in parallel.

formance scaling and depth/alpha acquisition in DVI graphics clusters. In *Proc. Workshop on Commodity-Based Visualization Clusters CCViz02*, 2002.

[13] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. Wiregl: a scalable graphics system for clusters. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques(SIGGRAPH)*, pages 129–140, 2001.

[14] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. D. Kirchner, and J. T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pages 693–702, 2002.

[15] D. Kirk. Innovation in graphics technology. *Talk in Canadian Undergraduate Technology Conference*, 2004.

[16] J. Krüger and R. Westermann. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Trans. Graph. (SIGGRAPH)*, 22(3):908–916, 2003.

[17] P. Lallemand and L. Luo. Theory of the lattice Boltzmann method: Accoustic and thermal properties in two and three dimensions. *Phys. Rev. E*, 68(036706), 2003.

[18] W. Li, X. Wei, and A. Kaufman. Implementing lattice Boltzmann computation on graphics hardware. *Visual Computer*, 19(7-8):444–456, December 2003.

[19] C. P. Lowe and S. Succi. *Go-with-the-flow lattice Boltzmann methods for tracer dynamics*, chapter 9. Lecture Notes in Physics. Springer-Verlag, 2002.

[20] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: a system for programming graphics hardware in a C-like language. *ACM Trans. Graph. (SIGGRAPH)*, 22(3):896–907, 2003.

[21] N. Martys, J. Hagedorn, D. Goujon, and J. Devaney. Large scale simulations of single and multi-component flow in porous media. *Proceedings of The International Symposium on Optical Science, Engineering, and Instrumentation*, June 1999.

[22] F. Massaioli and G. Amati. Optimization and scaling of an OpenMP LBM code on IBM SP nodes. *Scicomp06 Talk*, August 2002.

[23] F. Massaioli and G. Amati. Performance portability of a lattice Boltzmann code. *Scicomp09 Talk*, March 2004.

[24] R. Mei, W. Shyy, D. Yu, and L. S. Luo. Lattice Boltzmann method for 3-D flows with curved boundary. *J. Comput. Phys.*, 161:680–699, March 2000.

[25] L. Moll, A. Heirich, and M. Shand. Sepia: scalable 3D compositing using PCI pamette. In *Proc. IEEE Symposium on Field Programmable Custom Computing Machines*, pages 146–155, April 1999.

[26] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Numerical Mathematics and Scientific Computation. Oxford University Press, 2001.

[27] A. T. C. Tam and C.-L. Wang. Contention-aware communication schedule for high-speed communication. *Cluster Computing*, (4), 2003.

[28] C. J. Thompson, S. Hahn, and M. Oskin. Using modern graphics architectures for general-purpose computing: A framework and analysis. *International Symposium on Microarchitecture (MICRO)*, November 2002.

[29] S. Venkatasubramanian. The graphics card as a stream computer. *SIGMOD Workshop on Management and Processing of Massive Data*, June 2003.

[30] A. Wilen, J. Schade, and R. Thornburg. *Introduction to PCI Express*: A Hardware and Software Developer's Guide*. 2003.

[31] D. A. Wolf-Gladrow. *Lattice Gas Cellular Automata and Lattice Boltzmann Models: an Introduction*. Springer-Verlag, 2000.

[32] F. Zara, F. Faure, and J-M. Vincent. Physical cloth simulation on a PC cluster. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, pages 105–112, 2002.