MPIglut: Powerwall Programming made Easier

Dr. Orion Sky Lawlor
U. Alaska Fairbanks
http://lawlor.cs.uaf.edu/
WSCG '08, 2008-02-05
Talk Overview

• MPIglut: lets serial glut OpenGL apps run in parallel atop MPI
  ▪ Powerwall Hardware & Software
  ▪ Parallel Rendering Software
• MPIglut code & runtime changes
• Application Performance
• Future work
MPIglut Basics
MPIglut: Motivation

• Modern computing is parallel
  □ Multi-Core CPUs, Clusters
    • Athlon 64 X2, Intel Core2 Duo
  □ Multiple Multi-Unit GPUs
    • nVidia SLI, ATI CrossFire
  □ Multiple Displays

• But languages and many existing applications are sequential
  □ Software problem: run existing serial code on a parallel machine
  □ Related: easily write parallel code
What is a “Powerwall”?  

• A powerwall has:  
  ▪ Several physical display devices  
  ▪ One large virtual screen  
  ▪ I.E. “parallel screens”  

• UAF CS/Bioinformatics Powerwall  
  ▪ Twenty LCD panels  
  ▪ 9000 x 4500 pixels combined resolution  
  ▪ 35+ Megapixels
Sequential OpenGL Application
Parallel Powerwall Application
MPIglut: The basic idea

- Users compile their OpenGL/glut application using MPIglut, and it “just works” on the powerwall.
- MPIglut's version of glutInit runs a separate copy of the application for each powerwall screen.
- MPIglut intercepts glutInit, glViewport, and broadcasts user events over the network.
- MPIglut's glViewport shifts to render only the local screen.
MPIglut uses MPI parallel library

• **MPI**: Message Passing Interface
  - Standardized communication library for distributed-memory parallel machines (like clusters)

• MPI runs over many networks; several software implementations
  - MPICH, OpenMPI, AMPI

• MPIglut uses MPI to compile (mpiCC), start-up (mpirun), event broadcast, and synchronization
  - MPIglut apps can call MPI too!
MPIglut uses glut sequential code

- GL Utilities Toolkit
  - Portable window, event, and GUI functionality for OpenGL apps
  - De facto standard for small apps
  - Several implementations: Mark Kilgard original, FreeGLUT, ...
  - Totally sequential library, until now!

- MPIglut intercepts several calls
  - But many calls still unmodified
  - We run on a patched freeglut 2.4
    - Minor modification to window creation
Parallel Rendering Taxonomy

- Molnar's influential 1994 paper
  - Sort-first: send geometry across network before rasterization (GLX/DMX, Chromium)
  - Sort-middle: send scanlines across network during rasterization
  - Sort-last: send rendered pixels across the network after rendering (IBM's Scalable Graphics Engine, ATI CrossFire)
Parallel Rendering Taxonomy

• Expanded taxonomy:
  - Send-event (MPIglut, VR Juggler)
    • Send only user events (mouse clicks, keypresses). Just kilobytes/sec!
  - Send-database
    • Send application-level primitives, like terrain model. Can cache/replicate data!
  - Send-geometry (Molnar sort-first)
  - Send-scanlines (Molnar sort-middle)
  - Send-pixels (Molnar sort-last)
Code & Runtime Changes
#include <GL/glut.h>
void display(void) {
    glBegin(GL_TRIANGLES); ... glEnd();
    glutSwapBuffers();
}

void reshape(int x_size,int y_size) {
    glViewport(0,0,x_size,y_size);
    glLoadIdentity();
    gluLookAt(...);
}

...

int main(int argc,char *argv[]) {
    glutInit(&argc,argv);
    glutCreateWindow(“Ello!”);
    glutMouseFunc(...);
    ...
}

MPIglut Conversion: Original Code
MPIglut: Required Code Changes

```c
#include <GL/mpi glut.h>

void display(void) {
    glBegin(GL_TRIANGLES); ... glEnd();
    glutSwapBuffers();
}

void reshape(int x_size,int y_size) {
    glViewport(0,0,x_size,y_size);
    glLoadIdentity();
    gluLookAt(...);
}

... int main(int argc,char *argv[]) {
    glutInit(&argc,argv);
    glutCreateWindow(“Ello!”);
    glutMouseFunc(...);
    ... }
```

This is the **only** source change. Or, you can just copy mpiglut.h over your old glut.h header!
MPIglut Runtime Changes: Init

```c
#include <GL/mpiglut.h>

void display(void) {
    glBegin(GL_TRIANGLES); ... glEnd();
    glutSwapBuffers();
}

void reshape(int x_size, int y_size) {
    glViewport(0, 0, x_size, y_size);
    glLoadIdentity();
    gluLookAt(...);
}

int main(int argc, char *argv[]) {
    glutInit(&argc, argv);
    glutCreateWindow("Ello!");
    glutMouseFunc(...);
    ...
    return 0;
}
```

**MPIglut** starts a separate copy of the program (a “backend”) to drive each powerwall screen.
Mouse and other user input events are collected and sent across the network. Each backend gets identical user events (collective delivery)
MPIglut Runtime Changes: Sync

```c
#include <GL/mpi glut.h>
void display(void) {
    glBegin(GL_TRIANGLES); ... glEnd();
    glutSwapBuffers();
}
void reshape(int x_size,int y_size) {
    glViewport(0,0,x_size,y_size);
    glLoadIdentity();
    gluLookAt(...);
}
... int main(int argc,char *argv[]) {
    glutInit(&argc,argv);
    glutCreateWindow("Ello!");
    glutMouseFunc(...);
    ...
    }
```

Frame display is (optionally) synchronized across the cluster.
MPIglut Runtime Changes: Coords

```
#include <GL/mpiglut.h>
void display(void) {
    glBegin(GL_TRIANGLES); ... glEnd();
    glutSwapBuffers();
}
void reshape(int x_size,int y_size) {
    glViewport(0,0,x_size,y_size);
    glLoadIdentity();
    gluLookAt(...);
}
... int main(int argc,char *argv[]) {
    glutInit(&argc,argv);
    glutCreateWindow("Ello!");
    glutMouseFunc(...);
    ...}
```

User code works only in global coordinates, but MPIglut adjusts OpenGL's projection matrix to render only the local screen.
MPIglut Runtime Non-Changes

MPIglut does **NOT** intercept or interfere with rendering calls, so programmable shaders, vertex buffer objects, framebuffer objects, etc all run at full performance.
• Each backend app must be able to render its part of its screen
  ▪ Does not automatically imply a replicated database, if application uses matrix-based view culling

• Backend GUI events (redraws, window changes) are collective
  ▪ All backends must stay in synch
  ▪ Automatic for applications that are deterministic function of events
    • Non-synchronized: files, network, time
MPIglut: Bottom Line

• Tiny source code change
• Parallelism hidden inside MPIglut
  - Application still “feels” sequential!
• Fairly major runtime changes
  - Multiple synchronized backends running in parallel
  - User input events communicated across network
  - OpenGL rendering coordinate system adjusted per-backend
• But rendering calls are left alone
Delivered Application Performance
Performance Testing

- MPIglut programs perform about the same on 20 screens as they do on 1 screen
- We compared performance against two other packages for running unmodified OpenGL apps:
  - DMX: OpenGL GLX protocol interception and replication (MPIglut gets screen sizes via DMX)
  - Chromium: libgl OpenGL rendering call interception and routing
Benchmark Applications

UAF CS Bioinformatics Powerwall
Switched Gigabit Ethernet Interconnect
10 Dual-Core 2GB Linux Machines:
7 nVidia QuadroFX 3450
3 nVidia QuadroFX 1400

basic
mandel

tex, tex_obj
vtx, vtx_obj

soar
MPIglut Performance

Delivered Performance (fps)

Number of Screens and CPUs (MPIglut)

- basic
- tex_obj
- vtx_obj
- tex
- vtx
- soar
- mandel
Chromium Tilesort Performance

Gigabit Ethernet Network Saturated!
Conclusion & Future Work

• MPIglut: an easy route to high-performance parallel rendering

• Hiding parallelism inside a library is a broadly-applicable technique
  ▶ THREADirectX? OpenMPQt?

• Still much work to do:
  ▶ Multicore / multi-GPU support
  ▶ Need better GPGPU support (tiles, ghost edges, load balancing)
  ▶ Need load balancing, possibly by moving inter-processor boundaries
Backup Slides
What is a “Powerwall”?

• Powerwalls are often driven by a small parallel cluster
  ▶ Distributed-memory parallel machines
  ▶ Software must run over slow commodity network, often gigabit ethernet

• Porting existing software to powerwalls is a big problem!