

GPGP: Assignment 1 (Basic Linear Algebra on the GPU)

Date handed out: Feb 27, 2007

Due date: Mar 9, 2007

This assignment shall introduce you to (old style) GP computing using GPUs. It involves two simple linear algebra operations to be implemented on the GPU. The preferred implementation language is OpenGL + Cg/GLSL. You are welcome to try other languages (DX+HLSL, ARB assembly). Make a password protected webpage of your submission and email BOTH the instructors with the URL/password. You can use sample source presented in the classes (Intro to OpenGL by Jeremy, or Distance Computations) to get started.

Problem 1: saxpy(): For two vectors \mathbf{x} and \mathbf{y} of length N and a scalar value $alpha$, we want to compute a scaled vector-vector addition: $\mathbf{y} = \mathbf{y} + alpha * \mathbf{x}$. Load \mathbf{x} and \mathbf{y} as textures on the GPU. Readback the result to the CPU. Compare accuracy of your GPU implementation with a CPU implementation.

Problem 2: Dense Matrix-Vector multiplication: Given a matrix \mathbf{M} of size m times n and a vector \mathbf{v} of length n , compute $\mathbf{M}\mathbf{v}$. Exercises:

1. Try different sizes of m , n from 4 - 2K
2. Try different representations of \mathbf{M} and \mathbf{v}
3. Try different reduction techniques

Performance measurement: For each problem, benchmark your code performance wrt the following metrics:

1. Transform throughput: Verts/sec
2. Bus Transfer: CPU \leftrightarrow GPU data transfers (in GB/s)
3. Fragment throughput: Texels/sec
4. Compute throughput: FLOPs
5. Memory bandwidth: GB/s

Provide detailed explanations how you performed the benchmark measurements. Report which implementation (data representation, input size, reduction technique) gives highest performance for each metric. Why is this so? This will help you identify the bottlenecks in the GPU pipeline.

Extra credit.

Multiple saxpy(): Repeat problem 1, with 128 varying values of $alpha$, $alpha = 0, 1, \dots, 127$.

Sparse-matrix vector multiplication. Repeat problem 2, with following differences:

1. \mathbf{v} is sparse
2. \mathbf{M} is sparse
3. Both \mathbf{M} and \mathbf{v} are sparse.

A surprise prize will be given for the most efficient implementation (incl extra credit).