Bisman Walia

About me

I'm an undergrad studying computer science. Specific areas of interest include algorithms, artificial intelligence, and parallelism. I like strategy games and problem solving.

I would like to learn some non-obvious techniques for parallelizing algorithms - stuff that is more challenging than data parallelism. I am more interested in learning about how to parallelize algorithms than in learning how to map them to specific machine architectures.

One Application Problem

Parallel computing has been used to enhance real-time graphics rendering. This is commonplace in the world of 3D video games. This application spurred the development of extremely powerful and highly parallel hardware known as GPUs.

The basic problem is to draw huge numbers of polygons on a high resolution screen many times per second. This basically boils down to rendering billions of triangles. APIs such as OpenGL and DirectX were developed to interface with the GPUs.

This is one of the most successful parallel applications allowing hundreds of times better performance than what could be achieved with CPU-based code. This application also scales very well to hundreds or thousands of processors.

There are many layers to this application, so with optimized drivers for specific hardware, higher level applications like games can scale to arbitrary hardware with great performance improvements.

Traditionally, graphics rendering has been limited by the speed because people could construct worlds with great detail but too much processing power was required to render it. Therefore, textures and meshes had to be reduced to fewer and fewer polygons until these interactive applications could achieve a steady framerate.

Now the machines are powerful enough to render worlds with detail that is good enough for the displays people generally use so there is an emphasis on finding different ways to draw the environments. Different rendering techniques implemented in software, such as raytracing, have also been parallelized.