THE FUTURE OF PARALLEL COMPUTING: GPU VS CELL

General Purpose Planning against Fast Graphical Computation Architectures, which is the Best Solution for General Purposes Computation?

Luca Bianchi, Riccardo Gatti and Luca Lombardi
Department of Informatics Engineering, University of Pavia, Via Ferrata 1, Pavia, Italy

Keywords: Real-time Visual Simulations, Parallel Computing, HPC, CELL, GPU, GPGPU, General Purposes, Fast Computing.

Abstract: Complex models require high performance computing (HPC) which means Parallel Computing. That is a fact. The question we try to address in this paper is "which is the best suitable solution for HPC contexts such as rendering? Will it be possible to use it in General Purpose elaborations?" We start from these questions and analyze two different approaches, IBM CELL and the well known GPGPU, showing how changing our minds and breaking some assumptions can lead to unexpected results and open a whole set of new possibilities. We talk about rendering, but quickly move slightly towards general purpose computation, because many algorithms used in Visual Simulations are not only referred to rendering issues but to a wider range of problems.

1 HIGH PERFORMANCE COMPUTING

In a couple of years the demand for fast computation solutions has grown up dramatically, thanks to the wide diffusion of multimedia applications and the availability of complex models for visual simulations.

Many attempts have been done to push computer's CPUs to overcome their limits using the performance grow rate described by Moore's law. Until recently the performance gains for processors performances were obtained through clock frequency increasing. However this led to the rise of many unavoidable problems which slew down CPU growth rate, due to heating problems originating from too high an integration of circuits on a single chip.

A couple of years ago some solutions started to be implemented to overcome this strong limit which would have bounded elaboration performances and, indirectly, technology development. One interesting solution comes from 3D graphics and involves the adoption of parallel architectures for High Performance Computing. This gained wide success in the critical performance context of rendering for 3D Graphics. Certainly this solution was optimal and researchers soon started to investigate out of the rendering context in which it was initially developed. In this way a new paradigm emerged and was established: General Purpose on GPU (Luebke & Harris, 2004).

In the same years Sony, IBM and Toshiba started to develop a new processor architecture designed for parallel computing which was called CELL.

1.1 The Heart of the Problem

Many publications have been written to show how both solutions are better than traditional common CPUs. No one though has yet taken a position on how parallel computing will be in the next years.

We started by considering that Real-Time Visual Simulations often require to address problems not only concerned to rendering itself (e.g. physics, collision detections, artificial intelligence), and require fast computation even for general purpose algorithms. This is the reason why hardware producers, such as nVidia, try to propose their solution not only for renderings or graphics-related contexts.

Bianchi L., Gatti R. and Lombardi L. (2008).
THE FUTURE OF PARALLEL COMPUTING: GPU VS CELL - General Purpose Planning against Fast Graphical Computation Architectures, which is the Best Solution for General Purposes Computation?.
In *Proceedings of the Third International Conference on Computer Graphics Theory and Applications*, pages 419-425
DOI: 10.5220/0001099904190425

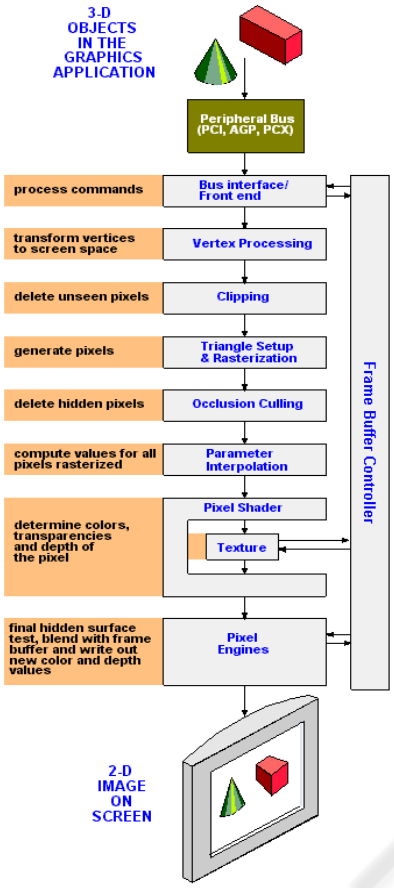

Figure 1: Standard Rendering Pipeline (The Computer Language Co. Inc., 2004).

With this fact clear in mind we asked ourselves i) which one between GPGPU and CELL is the best approach for HPC, ii) how much flexibility is important and iii) how much it is paid with performance loss.

Our goal is to show that CELL processor provides higher flexibility than the GPGPU approach as well as performances that are comparable if not better in many general purpose contexts. Our intention is not to run after clock frequencies, but to discuss how architectural choices make CELL a better solution for HPC. For this reason, we just present a small amount of data and then discuss them.

The next section introduces both the GPGPU and CELL approaches to parallel computing, assuming that both provide better performances compared to traditional CPUs (nVidia, 2006) (IBM, 2007).

Section 3 counters some common assumptions hinting that GPGPU could be the only solution for parallel computing and explains how CELL can easily overcome such assumptions.

Section 4 shows some of the benefits provided by this processor not available for GPGPU. Section 5, finally, presents the work we propose to unleash the power of CELL processor.

2 PARALLEL COMPUTING

2.1 General Purpose on GPU

This approach develops in the specific context of real – time 3D graphics out of the strong need for high accuracy with complex shading models and real – time performances for renderings.

A lot has been done since the late 90s when the first programmable unit for GPU was realized. Since then, the graphical pipeline was thought of not as a fixed sequence of steps built on GPU, but as a solution with two programmable stages (figure 1). This idea was supported by the adoption of a stream processing SIMD architecture which could perform a single instruction on multiple data in a single execution step (Akenine-Moller & Haines, 2002).

It is well known today which advantages in performance can be achieved through a programmable rendering pipeline, and not only at the graphic level. The idea behind the so called GPGPU is to play a trick by using the graphical unit for general elaborations. With this purpose, only the fragment shader is generally used: a single quad is sent down the pipeline, while data are stored in a texture as pixels' values. At rastering stage the quad is mapped on to screen dimension and a grid is built. Thanks to this object we can reference each data in our texture and use it for computation. Results are stored in a FrameBuffer from where they can easily be transferred back to CPU or sent to the screen. The program in the fragment shader could be any kind of algorithm, thus realizing general purpose computing (Luebke & Harris, 2004). What should be kept in mind is that GPGPU execution is still constrained by the concept of pipeline and computing is done using graphic hardware as a black box. One of the greatest limits to this approach is that each stream processor should work alone without sharing data from another processor.

2.1.1 Computer Unified Device Architecture and Tesla

In late 2006 nVidia Corporation sold the first GeForce 8800 card which used a new kind of pipeline (nVidia Technical Brief, 2006). This architecture, according to Shader Model 4 specs, can

provide more flexibility than the previous one. The separation between vertex and fragment units was left aside in moving to a new unified unit capable of load balancing (figure 2).

Also data transfers between stream processors were made more flexible by introducing a parallel data cache which could be used for making computed data available for other units. This is a great step forward, but GPU architecture still remains constrained by the pipeline paradigm, which has proved to be a good solution for fast renderings.

Recently, nVidia started proposing a modified version of GeForce 8800 named Tesla, with more memory and no DVI connectors (nVidia, 2007). This is a solution thought for GPGPU. Unfortunately, no data could be retrieved either for Tesla or for 8800 series. According to nVidia, CUDA graphic cards are 30% faster than other cards: as a result a comparison can be drawn even without data.

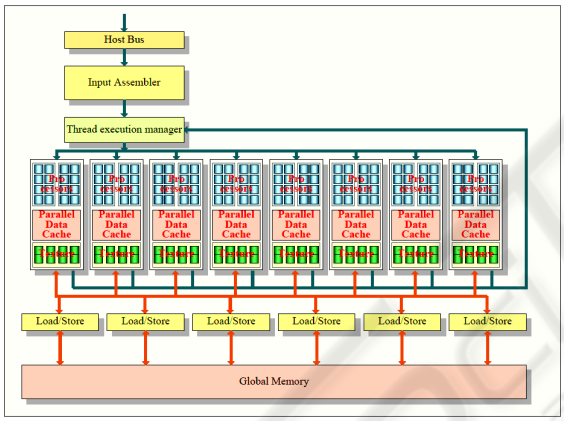

Figure 2: CUDA Architecture.

2.2 CELL Processor

CELL was designed by a partnership of Sony, Toshiba and IBM to be the heart of Sony's Playstation3 gaming console. However, results from Berkeley laboratory show that CELL architecture has a tremendous potential for scientific computations in terms of both raw performance and power efficiency (Williams, Shalf, Oliker, Kamil, Husbands, & Yelick, 2006).

CELL combines the considerable floating point resources for demanding numerical algorithms with a power-efficient software-controlled memory hierarchy. Instead of slowly evolving towards a streaming SIMD multi-core architecture, the CELL Processor was designed with these concepts in mind.

CELL architecture is made of nine processors operating on a shared, coherent memory. The processors can be distinguished between PPE and SPEs. PPE (PowerPC Processor Element) is a high performance 64-bit PowerPC core with 32KB L1 cache and 512KB L2 cache (figure 3).

Each SPE (Synergistic Processor Element) has a Synergistic Processor Unit (SPU), a 256KB local memory and a memory flow controller. SPEs are independent processors that are optimized for running compute-intensive applications.

PPE provides support for the operating system and manages the work of all the SPEs. PPE uses 2-way symmetric multithreading which is comparable to Intel Hyperthreading .SPEs, on the other hand, provide to CELL the application performance. Each SPE includes four single precision (SP) datapaths and one double precision (DP) datapath. SIMD double-precision operation must be serialized. The SPE cannot access the main shared memory and it must transfer data via DMA to its own local store using the Memory Flow Controller (MFC). The MFC operates asynchronously with respect to the SPU, so that is possible to overlap DMA transfers with other concurrent operations.

All CELL elements are connected by 4 data rings known as the EIB (Element Interconnection Bus). This ring permits 8 byte/s to be read and simultaneous transfers to be carried out.
Access to external memory is made by a 25.6 GB/s XDR memory controller.

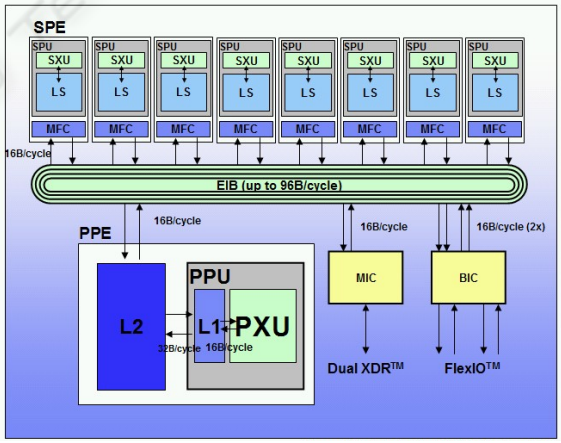

Figure 3: CELL Architecture Diagram (Gschwind, 2005).

3 COMMON BELIEFS

Thanks to game industry GPU parallel computing model is more widespread than CELL. This led to a series of assumptions that are often GPU biased. In

this section we present the most common claims and try to centre the balance.

3.1 GPUs do not Cost as Much as CELL

This first claim rises from one of the few papers available that draw a direct comparison between the two architectures (Baker, Gokhale, & Tripp, 2007). In this work it is shown how GPU has a lower price and an higher Speedup/$K rate. This could may appear obvious reading the paper, but some things need to be pointed out.

First of all, by looking to raw data the difference in performance and costs is not so huge because although CELL price is three times nVidia 7900 GTX price, it is also three times faster. In fact the Speedup/$K rate is almost the same: the difference is 0.34. The important thing is that in this benchmark a single graphic card vs a CELL blade system which mounts two processors is used. In order to make results comparable, only one Blade's processor is used. In this way we have the cost of a blade but half the power it could provide. Using the single CELL of a PS3 we discover that a single video card has the same price of a Playstation, which does not only includes the CELL processor. This small difference in terms of price is more evident if we compare the nVidia Deskside Tesla (sold at 7500$) and CELL QS21 Blade (almost 8000$). What Baker, Gokhale & Tripp (Baker, Gokhale, & Tripp, 2007) show is the importance of the possibility to buy a single CELL solution without all the PS3 environment.

The last thing to point out is that in the paper the code used for benchmarks is not optimized. This affects more the CELL performances than the GPU's, as we will discuss further on.

3.2 GPUs have a Faster Learning Curve

As a matter of fact GPUs have a faster learning curve if your aim is just to write a "Hello world" program. If your goal is to use GPU for small algorithms with no high performance needs you will be able to do that after a while. On the contrary if your goal is to develop an optimized solution for a problem where performances really matter, then you will have to learn graphic programming and OpenGL (or DirectX). This will not make your learning curve so fast. Some good news comes from nVidia with the announcement that a C compiler will be available for CUDA. In this way learning graphics will be no longer necessary but you will always need to know how your code is executed on GPU. This is the very problem which makes the CELL the learning curve so slow.

In considering learning curves, the only difference worth pointing out is that GPU makes parallel programming transparent to users (nVidia CUDA, 2007). However it has not yet been demonstrated that this would be and advantage in specific contexts where optimizations matter.

3.3 GPUs are Specific for Graphic and Provide Better Performances

This is a claim often proposed while presenting benchmarks between GPUs and CPUs, and is obviously true. If you have an algorithm, the closer it is to graphic context, the more porting it to GPU would provide faster performance. The common example is image filtering, where we can obtain an incredible speedup with respect to CPU implementations. What is never said but often thought is that GPU performances are the best tool available in parallel computing, both at the graphic and general purpose levels. This is not true.

One of the most significant results provided by CELL over GPGPU architecture concerns the solution of a matrix multiplication problems. This has been used for a long time to demonstrate the GPU's abilities. nVidia Quadro 4600 performs single precision matrix multiplications with a throughput of 90 GFLOPS (GPU-Tech, 2007). The same operation performed on CELL processor with 8 SPU runs at 140 GFLOPS (Barcelona Supercomputing Center, 2007). This result is highly significant, as matrix multiplication has always been GPU computing's greatest achievement. We do not aim to claim that CELL should be used for graphics rendering. Our purpose is just to demonstrate that, if this processor is valuable even for a context where GPU has always been the top solution, its flexibility probably makes it a better choice for general purpose parallel computing.

It might be argued that, on paper, nVidia G80 offers a higher GFLOPS rate than CELL (500 against 208). This claim is true if you only compare the raw computation rates, positing a full utilization of both technologies. This is just an ideal case. In real applications, code optimization is extremely difficult for GPU, and is even more so if we consider the C compiler layer introduced by CUDA architecture. In poor words in real applications, such as real-time ray tracing, CELL benefits from code optimization more than GPU and provides higher performance even with the single six core CELL

processor of Playstation 3 as shown in (Minor, 2007). In this paper, both architectures are compared, first on raw computation GFLOPS and then with the graphics algorithm of Interactive Raytrace applied to Stanford Bunny. The results are amazing: one single CELL processor is four to five times faster than G80. If we consider the "on paper" computing power and use a QS20 blade (which has a comparable GFLOPS amount), it is eight to eleven time faster.

4 WHY A GOOD SOLUTION IS A GOOD SOLUTION

The CELL architecture provides features that makes it an excellent platform for developing any kind of applications. We identify two main benefit in CELL structure: the possibility of using and organizing the work of the different cores in a totally separate and independent way and the fast communication system that link all the chip components. A developers framework for CELL offers useful tools like profilers, simulators and compilers for helping the programmers to take advantage of all the CELL key features.

4.1 Flexibility

The CELL Broadband Engine Architecture has been designed to support a variety of different applications.

Although the CELL processor was initially conceived for application in game consoles or high-definition televisions, its architecture was designed to allow fundamental advances in processor performance and programming flexibility.

The GPU Architecture, on the other hand, was initially designed as a dedicated rendering device and is highly efficient in making more effective all those algorithms and all those computations bound to graphics needs. Using a dedicated architecture to make general purpose applications requires the programmer to deal with a large number of problems and limitations in their algorithms. In fact, the GPGPU concept of programming is based on deceiving the GPU by using the graphics pipeline for making different types of computations unrelated to graphics applications.

Programs that run on CELL typically split computational cost among all the available processor elements. In order to determine workload and data distribution, the programmer should take the following considerations into account:

- Processing-load distribution
- Program structure
- Data access patterns
- Code movement and data movement among processors
- Cost of bus loading and bus attachments

In the CELL programming way there are different application partitioning models can be found. The two main models are the PPE-centric model and the SPE-centric model.

In the PPE-centric model the main application runs on the PPE while the SPEs are used to off-load other individual tasks. The PPE duties are to wait and coordinate the different results coming from the SPEs. Applications that have serial data and parallel computations fit this model well. The SPEs can be used in three different ways:

- The multistage pipeline model
- The parallel stages model
- The services model

If an application requires multiple and sequential stages, the programmer can use a multistage-pipeline model approach. Every step of the application is loaded onto a single SPE and the results are sent through the shared bus from SPE to SPE. The data stream is initially sent to the first SPE and the results can be taken from the last SPE that contains the last stage of the application. In Multistage pipelining problems occur in determining load balancing and in large data-movement between the SPEs.

In the parallel stages model each SPE runs the same task and the data input of the application is equally split among all the SPEs as well as processed at the same time. This is a concept of programming similar to the GPGPU where the input data stream is processed at the same time in different shaders running the same kernel.

The PPE-centric service model is used when there is the need to run different tasks that are part of a large application not in a pre-existing order. In each SPE a different program is loaded and the appropriate SPE is called by the PPE when a particular service is needed.

In the SPE-centric model the application code is split among all the SPEs (or part of them). Each SPE fetches its next work from either the main storage or its local memory. The PPE on the other hand acts as a resource manager for the SPEs.

All this flexibility in using the different CELL's cores makes it a perfect platform for any kind of application. The programmer just has to devise the best way to organize the steps of his algorithms to exploit all the possibilities and the power of the CELL architecture.

There are already many papers that show how CELL architecture boosts the performance of many kinds of applications ranging from rendering to general purpose ones. A work from Utah University shows how good the performances of ray tracing on the CELL Processor are. The research shows how to efficiently map the ray tracing algorithm to the CELL Processor, with the result that a single SPE attains the same performance as a fast x86 system. (Benthin, Wald, Scherbaum, & Friedrich, 2007). Another work shows how a parallelized form of H.264 encoding algorithm (Park & Soonhoi, 2007) achieves optimal performance. In this work the authors also claim that a SPE-specific optimization is needed to obtain a meaningful speed-up. By using the Vector/SIMD instructions and reducing data transfers between SPE and PPE, better performance can be achieved in their particular application.

Some effort were also made in porting a digital media indexing application (MARVEL) on CELL processor. This kind of application needs image analysis for feature extraction; overall performance of this algorithms was excellent on CELL platform (Lurng-Kuo, Qiang, Apostol, Kenneth, Smith, & Varbanescu, 2007).

Again, all these examples show how the CELL architecture is suitable for improving performance of a different range of applications.

4.2 Shared Memory

The CELL processor can be programmed as a shared-memory multiprocessor where SPE and PPE units can interoperate in a cache-coherent shared-memory programming model. Anyway PPE and SPE have significant difference in the way they access memory. PPE accesses main storage with load and store instructions that go between a private register file and main storage. SPE accesses main storage with direct memory access (DMA) commands that are stored, along with data, in a private local memory. This 3-level organization (register file, local store and main storage) explicitly parallelizes computation and the transfer of data and instructions. The main reason for this organization is that application performance is, in most cases, limited by memory latency rather than by peak compute capability or peak bandwidth. The DMA model allows each SPE to have many concurrent memory accesses. Another benefit is that very few cycles are needed to set up a DMA transfer compared to the long waiting time (in terms of cycles) that occurs when a load instruction of a program misses in the caches in conventional architecture.

A valid approach in memory-access is to create a list of DMA transfers in the SPE's local store so that the SPE's DMA controller can process this list asynchronously while the SPE operates in previously transferred data.

The on-chip communication benchmark of the CELL was matter of accurate benchmark and tests. Overall results of the experiments demonstrate that the CELL processor's communication subsystem is well matched to the processor's computational capacity. The communication network provides all the speed and bandwidth that applications need in order to exploit the processor's computational power (Kistler, Perrone, & Petrini, 2006).

4.3 Simulator, Compiler and Profiler

One of the main problems while programming GPGPU kernels is the portability of the code. There are many differences between architectures of different manufacturers that prevent the code to be freely used on any GPU (g.e texture format, texture size, pixel format supported …). On the other side CELL architecture provides to the programmer with a unique and complete environment.

A Full-System Simulator is offered as an alternative to conventional process and thread programming. Here the programmer has access to many features such as scheduler for threads, debugging tools, performance visualization, tracing and logging capabilities.

PPE implements an extended version of the PowerPC instruction set. This extension consists of a Vector/SIMD Multimedia extension plus some changes in PowerPC instructions. The SPE instruction set is similar to PPE but needs a different compiler. All these extensions are supported by C-language intrinsics. Intrinsics substitute assembly instructions with C-language commands. Most instructions process 128b operands, divided into four 32b words.

5 CONCLUSIONS

Both GPGPU and CELL approaches are excellent solutions for HPC applications. Without any doubt they will mark the state of the art for next years. Many upcoming changes will be released, starting from CELL v2.0 through next CUDA generation especially designed for physics.

Hopefully we've proved that CELL has the best opportunities to become the standard for general purposes computing: its flexibility could provide high performances without too many constraints.

An additional gain of CELL is represented by its reduced size which makes it suitable for embedded devices.

Our interest in the topic is focused on creating a good knowledge base for CELL programming, using it to reduce computational costs in general purpose contexts as Medical Image Elaboration or Virtual Reality. This lack of knowledge and realizations on CELL are today one of the biggest obstacles its adoption because strengthens the idea of a solution which don't pay the investment and with a too abrupt learning curve.

ACKNOWLEDGEMENTS

Special thanks to Elisa Ghia for all the support given to this work.

This work has been partially supported by PRIN 2006 - *Ambient Intelligence: event analysis, sensor, reconfiguration and multimodal interfaces.*

REFERENCES

Akenine-Moller, T., & Haines, E. (2002). *RealTime Rendering.* A. K. Peters.

Baker, Z. K., Gokhale, M. B., & Tripp, J. (2007). Matched Filter Computation on FPGA, Cell and GPU. *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines.*, (pp. 207-218).

Barcelona Supercomputing Center. (2007). *Matrix Multiplication Example*. Retrieved 11 25, 2007, from Computer Sciences: http://www.bsc.es/plantillaH.php?cat_id=420

Benthin, C., Wald, I., Scherbaum, M., & Friedrich, H. (2007). Ray Tracing on the Cell Processor. *IEEE Symposium on Interactive Ray Tracing* (pp. 15-23). IEEE.

GPU-Tech. (2007). *GPU-Tech GPU Computing*. Retrieved 11 26, 2007, from GPU-Tech: http://www.gpucomputing.eu/index3.php?lang=en&page=_demo1.php&id=2

Gschwind, M. (2005, 08 17). The Cell project at IBM Research.

IBM. (2007). *Cell Broadband Engine - An Introduction.*

Kistler, M., Perrone, M., & Petrini, F. (2006). Cell Multiprocessor Communication Network: Built for Speed. *IEEE Micro , 26* (3), 10-23.

Luebke, D., & Harris, M. (2004). GPGPU: General Pourpose Computation On Graphics Hardware. *SIGGRAPH Course Notes.*

Lurng-Kuo, L., Qiang, L., Apostol, N., Kenneth, R. A., Smith, J. R., & Varbanescu, L. A. (2007). Digital Media Indexing on the Cell Processor. *IEEE International Conference on Multimedia and Expo.* IEEE International.

Minor, B. (2007, 09 05). *Cell vs G80*. Retrieved 11 23, 2007, from Game Tomorrow: http://gametomorrow.com/blog/index.php/2007/09/05/cell-vs-g80/

nVidia CUDA. (2007). GeForce 8800 & NVIDIA CUDA A New Architecture for Computing on the GPU.

nVidia. (2006). *CUDA Programming GUIDE v0.8.* nVidia.

nVidia. (2007, 10 18). *nVidia Tesla Tech Specifications*. Retrieved 11 18, 2007, from nVidia Web site: http://www.nvidia.com/object/tesla_tech_specs.html

nVidia Technical Brief. (2006). *NVIDIA GeForce 8800 GPU Architecture Overview.* nVidia Corporaion.

Park, J., & Soonhoi, H. (2007). Performance Analysis of Parallel Execution of H.264 Encoder on the Cell Processor. *IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia, 2007* (pp. 27-32). ESTIMedia.

The Computer Language Co. Inc. (2004). Graphics Pipeline. Retrieved from Answer.com: http://www.answers.com/topic/graphics-pipeline?cat=technology

Williams, S., Shalf, J., Oliker, L., Kamil, S., Husbands, P., & Yelick, K. (2006). *The Potential of the Cell.* Berkeley : Computational Research Division Lawrence Berkeley National Laboratory.

Wright, R. J. (2004). *OpenGL SuperBible.* Addison Wesley.