Graphics and Multimedia
CS 203 Final Project
Ulises Amaya, Paul DiLorenzo, Anna Majkowska

Summary of "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications" [1]

By: Ulises Amaya

This paper exploits two types of hardware adaptations with the goal of saving energy in general purpose processors used in multimedia applications. The first type is named Architectural adaptation and the second is Dynamic Voltage (and frequency) Scaling or DVS. The paper develops an algorithm to examine both approaches individually as well as combined to see which is better and under which circumstances a specific approach would perform best.

The way multimedia applications process data is by dividing its work in units called frames. These frames must be processed in a finite amount of time named a deadline. If the processor finishes its task before the deadline then it has to wait for the next load to come and begin processing. This time is called a slack, which is the time that the processor remains idle. The aim of the hardware adaptations is to slow down the processor so that it remains the least amount of time idle but at the same time completes the work on time. If the processor works slower it requires less energy. The DVS approach reduces the voltage to reduce energy. The Architectural adaptation will reduce capacitance which reduces IPC and thus energy consumption.

There are some important assumptions that the algorithm makes:
"IPC is almost constant for different frames of the same type at a given frequency".
"IPC of a frame is almost independent of clock frequency since little time is spent in memory stalls"
"For a given frame type, instruction count varies slowly from frame to frame"
"Assume hardware recognizes when a new frame begins, type of frame, and its deadline"

There are two phases in the algorithm for discrete DVS. The first, called profiling phase is where it decides how much work can it be done meeting the deadline. It orders the hardware configurations in increasing order of energy per instruction (EPI). The second called adaptation phase will predict the next set of instructions for the next frame of the same type. These phases can be implemented by software or hardware.

Problems appear with continuous DVS systems such as the Intel Xscale processor. Modifications to the algorithm were made for this case which includes working with minimum and maximum frequencies that would meet the deadline and correlate it with the discrete DVS algorithm. If it misses a deadline in the discrete case it will update the IPC and power values for the frame that was completed. In the continuous case the IPC for the architecture is updated after the frame is computed.

The paper finds that most of the time architectural adaptation and DVS combined consume the least amount of energy but they also find that DVS alone is responsible for the greatest amount of energy savings being between 68-78% vs. 22% mean for the architectural adaptation. The latter alone gives a very good performance when a less aggressive architecture is chosen and it has a high number of instructions per frame. Otherwise, DVS will be better.

Thorough Part: Newest Low Power Consumption methods taken by the leading Graphic Processor developers, ATI and Nvidia

By: Ulises Amaya

These developers take an interesting approach in their newest technologies combining the software and hardware into a hybrid technology. They expand it and use it not only in the GPU but also in the main CPU itself as well as to other critical parts or the hardware in a given device. The idea is to have the best balance between performance and power consumption when the device is battery powered. Concerned with the high power consumption mainly by the primary CPU, GPU and Display in a mobile device such as a laptop, the PowerMizer technology from Nvidia™ controls different parts of the hardware form both the hardware itself and from software therefore constructing a hybrid technology. The CPU consumes more power than the GPU, so one approach that is taken is to offload sophisticated geometry and rendering calculations from the CPU and put them into the GPU. They also add a highly efficient hardware MPEG-2 decoder for DVD playback allowing the CPU to run at its lower possible power level. When dealing with the display, the software allows the user to set the limits of brightness and then the software will *Dim* the display as needed in order to save as much power as possible. Another technique used is the DVS approach, using dynamic clock and voltage scaling. The idea is to lower the frequency or scale the voltage at which the GPU operates in order to save energy as described previously in this report. The last approach taken by this technology is to use dynamic clock gating. If the clock gating is zero, then the power consumption is zero. This allows units in the GPU that are not needed to shut down. http://www.nvidia.com/object/feature_powermizer.html PowerMizer4_102803.pdf

POWERPLAY™ Technology from ATI extends the dynamic clock speeds changes to happen not only in the GPU but also in the video memory clock. When detecting that the device is battery powered the GPU core clock frequency and voltage is dynamically scaled by using a technology they call Power-On-Demand that acts upon user activity which is constantly monitored. Another technique used to save power is to lower the refresh rate in the LCD screens which are a major power consumption part of a laptop's hardware.

http://www.ati.com/products/pdf/powerplaywp2.pdf

Summary of "ZR: A 3D API Transparent Technology for Chunk Rendering" [2]

By: Anna Majkowska

Chunk rendering, known also as zone rendering (ZR), is a technique for reducing memory bandwidth and thus achieving better graphical performance. Chunk-based systems sort graphic primitives and states into *chunks*, which correspond to non-overlapping regions of the graphics display. Each chunk is then rendered separately, using on-die depth and color caches. In this way all reads and writes to color and depth buffers are performed on die, and only final results are written to the frame buffer in memory. This significantly reduces 3D graphic bandwidth compared to conventional rendering, where all read/write operations use the frame buffer.

Chunk rendering also allows efficient anti-aliasing with super-sampling technique and increases the accuracy of blending, by using on-die buffers with extended precision and higher resolution than the frame buffer in memory. To achieve even higher speed-up, multiple chunks can be processed in parallel. As shown in [2] chunk rendering requires fairly small changes in the conventional rendering pipeline.

Although the technique of chunk rendering has been known since late 80's, there remained two unsolved problems which the authors address in

their paper: compatibility with existing 3D APIs and display list footprint growth.

The compatibility issue is caused by traditional APIs allowing applications to directly write or read from the frame buffer during the process of generating graphic primitives. Chunk rendering requires the collection all states and primitives before rendering a scene.

As shown in the paper, direct writes to the frame buffer are fairly rare in typical graphics applications. Some of them, such as 2D blits (arrays of data), can be represented as 3D primitives and split into chunks along with other primitives. For events, which cannot be represented as primitives (such as buffer locks in order to read the frame buffer and modify it depending on the read value), the authors propose a *serialization* technique. When such an event reaches the front end of the pipeline, the pipeline is flushed and all chunks with primitives collected so far are rendered. After the direct write to the frame buffer is performed, chunk rendering mode is restored. This technique does not cause significant performance reduction because serializing events are rare.

The issue of growth of display list footprint was addressed by a series of optimizations in the chunk rendering algorithm. The authors concentrate on reducing state command replication in chunks, as it dominates over the issue of replicating primitives. Optimizations include placing state commands only in chunks which contain a corresponding primitive, passing frequently occurring command parameters as pointers and encoding groups of commands as an aggregate state command.

The authors performed pre-silicon simulations, which showed that with their implementation of chunk rendering, total memory bandwidth requirement is 2-3 times lower and 3D performance 2-3 times higher than with conventional pipelines (without chunk rendering). The cost of architecture modifications, including pipeline changes and increase of on-die caches is low compared to the performance benefits.

The described technology was implemented in the Intel 830 integrated chip set. The test, with use of 3DwinBench'00 and 3Dmark'00, showed that the real performance gains were close to the pre-silicon simulations.

As the authors have shown in the paper, their chunk rendering technique offers significant reduction of graphics memory bandwidth at relatively low cost.

Thorough Part: how chunk rendering is implemented in modern graphics processors [3]

By: Anna Majkowska

The technique of chunk rendering is implemented in the 845G chipset as a part of integrated Intel Extreme Graphics, and used on Pentium 4 platforms. The chipset uses 32 bits per pixel graphics and shares memory resources with the rest of the system, so reducing the memory bandwidth becomes an important issue.

The use of zone rendering technique eliminated the need for depth buffer reads and writes, by processing only a single chunk in an on-die cache. Now, that all depth calculations are done on-chip, there is no need for a separate depth buffer.

The lower memory bandwidth made it also possible to use a simpler memory technology and memory interface, which created substantial savings in memory cost.

Architecture with chunk rendering can work under higher fill rate requirements than traditional architecture. *Fill rate* is the number of pixels that can be drawn per unit of time. In traditional architectures in scenes with a high level of complexity each pixel will be redrawn many times, when objects in the scene overlap. In ZR systems all pixel operations are performed in fast cache, and the fill rate requirement is reduced to the number of the pixels in a scene.

The same technique is also used in PowerVR graphics processors with an additional optimization of grouping triangles into strips, which maximizes memory usage efficiency. Thanks to chunk rendering Hidden Space Removal is performed at high speed, as each line in a tile is processed in one clock cycle. All blending operations are performed in True Color (32 bits precision), while frame buffer has only 16 bits color depth.

The PowerVR MBX family includes MBX Lite processors used in handheld devices and mobile phones, and MBX Pro – for arcade machines and video game consoles.

Summary of "Enhancing loop buffering of media and telecommunications applications using low-overhead predication" [4]

By: Paul DiLorenzo

Most media and telecommunication focused processors use the VLIW design. The VLIW design is a good match with the media and telecommunication applications. Although in the embedded market, where most of the media and telecommunication markets are in, branch resolution and instruction fetch waste cycles. Previous algorithms such as branch predictors and instruction cache are too expensive for the embedded market. Therefore, these ideas were replaced with a loop buffer.

A loop buffer is a "software-controlled, straight-line-code with knowledge of counted loops." This allows certain portions of the code to be buffered into the loop buffer which ultimately provides a speedup. But, loop buffers cannot handle control flow since, like the definition above says, is designed as a straight-line-code. This allows loops buffers to hold only simple loops. Full predicate, as seen in Intel's Itanium processor, is a solution to this problem.

A full predicate implementation "allows the compiler to have general control without branches." The drawback to this solution is the high encoding cost which is unacceptable for embedded processors. Therefore in media and telecommunication processors they have implemented partial predicate. In partial predicate, there are additional instructions that allow conditional moves or allowing condition codes to guard some instructions for executing. The problem with this solution is that they do not allow if-conversion.

If-conversion makes "any acyclic region of control flow into an equivalent single-entry straight-line segment of code", by converting the code into predicated code. But the problem with if-conversion is that it does not work with nested-loops. This presents a problem in the media and telecommunication area where matrix operations almost always come in the form of a nested loop.

In this paper, they present a compiler that converts nested loops and makes use of if-conversion techniques to enable 70 – 99% of operations to come from a statically managed 256-instruction loop buffer. To complement this improvement, they devised a predicate model that will allow general if-conversion. This improvement only takes a single bit to encode which is a vast improvement in hardware overhead compared to a full predicate solution.

Putting It All Together: How described techniques can be used in video game systems

By: Ulises Amaya, Paul DiLorenzo, Anna Majkowska

Chunk rendering is a technique that could prove to be useful in the video game console area. The problem with this idea is that the chunk rendering technique does not take into account the problems and obstacles that are inherent in the embedded market, such as cost and energy while maintaining performance. We present two methods that merge chunk rendering with two embedded techniques to attack the cost

and energy problem in video game and mobile game systems.

In video game systems such as Sony's Playstation 2 and Nintendo's Gamecube, cost is an important factor. Consoles must be inexpensive to attract buyers and thereby build a large market for games, which bring the bulk of the profit. Chunk rendering can be introduced without significant increase in the cost of the processor, as required changes in the pipeline are fairly small.

In computer graphics, including the chunk rendering technique, loops are very common. The inherent problem with loops is the wasted cycles in branch resolution and instruction fetch. Previously, we described a technique for embedded systems to convert loops into straight-line code that can be used inside loop buffers instead of more costly techniques such as branch predictors and instruction cache. By applying this method to the chunk rendering technique, we are able to provide performance boosts that video game enthusiasts demand while keeping the cost down.

An additional problem arises in mobile video game systems such as Nintendo's Gameboy. In this arena cost is an important factor for the same reasons explained above. But a more important problem is energy, since these systems operate on batteries. Users of these systems do not want to go through batteries every hour.

Previously we described a technique to save energy by using two types of hardware adaptation: Architectural Adaptation and Dynamic Voltage Scaling (or DVS). By implementing these techniques, along with chunk rendering, we can improve the performance of the system and keep the energy usage at a reasonable level. A potential danger may be the increase in energy use as the on-die cache size grows and the lack of a sufficiently aggressive architecture to make the DVS adaptation perform best. However, the displays of handheld devices are fairly small, so chunks can be made smaller as well, which will reduce the need for large on-die caches. This will make chunk rendering more suitable for embedded systems by reducing the cost and energy of the handheld video game device. For the Architecture Adaptation, the architectural approach could prove very useful given that some architectures are less aggressive (i.e. low IPC) and this is when it performs best.

We believe that by combining techniques intended for embedded systems with those used in computer graphics we can create new solutions, which can improve efficiency and conserve energy in different types of video systems.

References:

[1] C.J. Hughes, J.Srinivasan and V. Adve, “Saving energy with architectural and frequency adaptations for multimedia applications”, MICRO 2001

[2] E. Hsieh, V. Pentkowski, T. Piazza, “ZR: A 3D API Transparent Technology for Chunk Rendering”, MICRO 2001

[3] Based on: “Zone Rendering White Paper”,
http://www.intel.com/support/graphics/intel845g/whitepaper.htm
and “PoverVR Tile-based rendering”,
http://www.pvrdev.com/pub/PC/doc/f/PowerVR%20Tile%20based%20rendering.htm

[3] J.W. Sias, H.C. Hunter, W.W. Hwu “Enhancing loop buffering of media and telecommunications applications using low-overhead predication”, MICRO 2001