Applications of Small Scale Reconfigurability to Graphics Processors

University of Virginia Computer Science Technical Report CS-2005-11

Kevin Dale, Jeremy W. Sheaffer, Vinu Vijay Kumar, David P. Luebke,
Greg Humphreys, and Kevin Skadron
{kdale, jws9c, vv6v, luebke, humper, skadron}@virginia.edu

Abstract

We explore the application of Small-Scale Reconfigurability (SSR) to graphics hardware. SSR is a relatively new architectural technique wherein functionality common to multiple subunits is reused rather than replicated, yielding high-performance reconfigurable hardware with reduced area requirements. We show that SSR can be used effectively in programmable graphics architectures to allow double-precision computation without affecting the performance of single-precision calculations and to increase fragment shader performance with a minimal impact on chip area.

1 Introduction

Every hardware system makes a tradeoff between performance and flexibility. At one end of the spectrum, general purpose processors provide maximum flexibility at the expense of performance, area, power consumption, and price. Custom ASICs are the other extreme, providing maximum performance at a minimum cost, albeit for only a very narrow set of applications.

Modern graphics hardware lies somewhere between these two extremes; very high performance is desired while maintaining the flexibility of a programmable processor. Traditional in-between hardware solutions like FPGAs are inappropriate for graphics processors because of their large size and low performance relative to their fixed-logic counterparts [12]. Small-scale reconfigurability (SSR) provides an attractive compromise; systems that use SSR components can approach the high speed and small size of ASICs while providing some specialized configurability. In this paper, we explore the applicability of SSR to programmable graphics hardware.

The simplest example of a reconfigurable component is two fully functional components connected with a multiplexer (see Figure 1). Although these two components are disjoint, in typical usage they will contain substantially similar redundant substructures, which is precisely the situation in which SSR performs best. Rather than replicate all of the redundant structure, we can instead create a single component with several internal multiplexers, replicating only that substructure for which additional latency would be acceptable to our application.

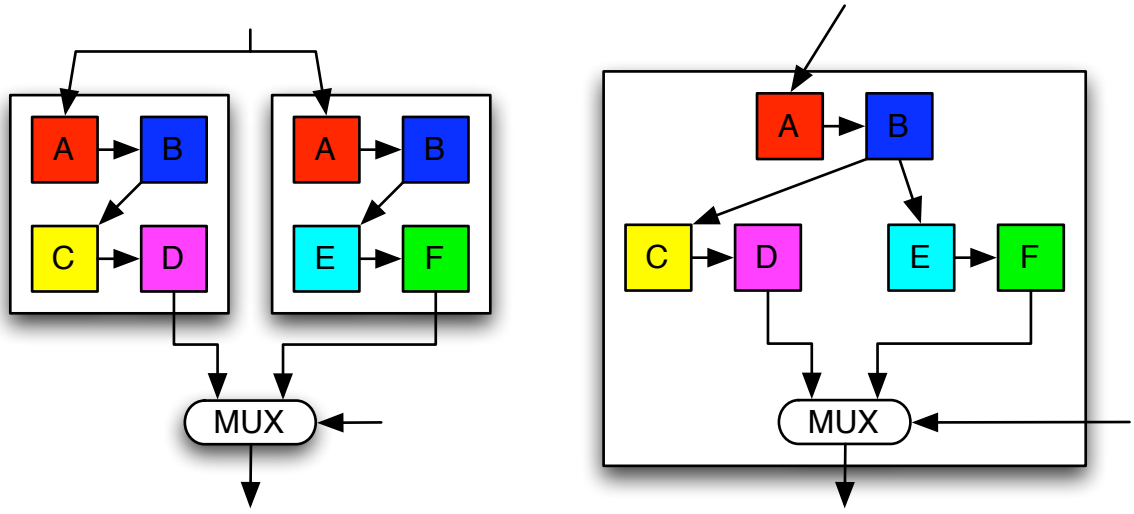
A common SSR unit is the morphable multiplier. These multiplier-adders can be reconfigured into a multiplier or an adder in a single cycle. When used to create single-precision floating point units, morphable multipliers yield a nearly 17% reduction in total area when compared to the sum of the sizes of their constituent parts [3].

Graphics processors, like specialized multimedia processors and DSPs, are a particularly suitable target for SSR due to their vector-processor like operations. When the same operation is performed repeatedly in SIMD fashion, reconfiguration and its associated overhead is infrequently needed, and any cost can be amortized over many instructions. Furthermore, SSR-based components typically have lower static power requirements because less hardware goes unused.

2 Related Work

Dynamically reconfigurable hardware has been a hot topic in recent computer architecture literature, especially in the FPGA and reconfigurable computing communities. The configurability of these systems serves myriad design goals, among them improved performance, power, area, and fault tolerance characteristics.

Even et al. describe a dual mode IEEE multiplier—a pipelined unit capable of producing one double-precision or two single-precision multiplications every clock cycle with a three cycle latency [5]. The authors argue that the reuse of substructure yields a cheap device that performs well for both precisions. They further claim that the single precision mode is particularly useful for SIMD applications, like graphics, because it is conducive to systems on which the



(a) Naïve reconfigurable hardware

(b) A more intelligent solution

Figure 1. A naïve implementation of reconfigurable hardware can be built by simply multiplexing between two distinct, unmodified units (left), but a more efficient design would reuse common substructure to avoid replication (right).

same operation is regularly repeated on large numbers of data points.

Guerra et al. explore *built-in-self-repair* (BISR) and its application to fault tolerance, manufacturability, and application specific programmable processor design [6]. Previous work in the area of dynamic repair had made use of specialized redundant units to replace damaged units; their paper describes the synthesis of more general units that can replace any of several units on a chip when damage is detected. The authors coin the term HBISR (*heterogeneous BISR*) for the technique.

A *morphable multiplier* is a device capable of performing either a floating point multiply or add using the same hardware structure [3]. Morphable multipliers require less area than the sum of the area needed for a separate multiplier and adder (in fact, they require only slightly more than a multiplier alone), while imposing negligible performance penalties.

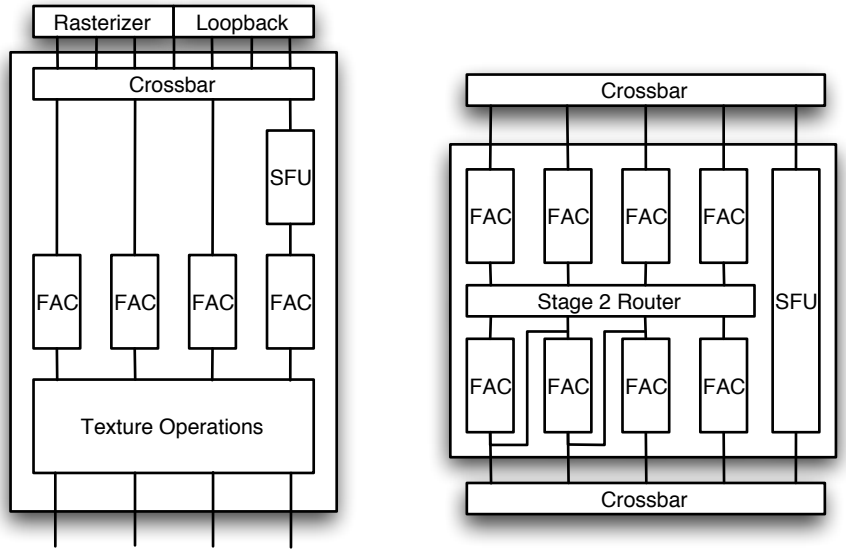
Metrics like area, performance, and power are easily quantified, but it is less obvious how to measure the increasingly important metric of hardware flexibility. Compton and Hauck have defined a testing method and quantification metric for flexibility of reconfigurable hardware [4]. Other examples of relevant research in reconfigurable hardware include Kim et al. [8] and Chiou et al. [2].

The work in this paper makes use of Brook [1], a stream-based programming language which allows the programmer to write general-purpose applications for a GPU without worrying about the sometimes byzantine details of GPU programming. Our experiments all use Chromium [7] to intercept and analyze streams of graphics commands made by real applications. The primary advantage of using Chromium is that we ensure that our workloads are not contrived. Although we use Brook and Chromium without modification, we have enhanced the Qsilver graphics architectural simulator [10, 11] to model the necessary aspects of the fragment pipeline. A detailed description of our modifications to QSilver and our experimental setup are presented in sections 3 and 4.

3 Simulation Setup

Qsilver is a simulation framework for graphics architectures that can simulate low-level GPU activity for any existing OpenGL application [10]. Qsilver uses Chromium [7] to intercept and transform an OpenGL application's API calls and create an annotated trace that encapsulates geometry, timing, and state information. This trace serves as input to the Qsilver simulator core, which performs an accurate timing simulation of the graphics hardware and produces detailed statistics.

Qsilver is configured at runtime with a description of its pipeline. In these experiments we simulate an NV4x-like architecture, with a pipeline configuration similar to that of



(a) Stage 1

(b) Stage 2

Figure 2. Proposed SSR fragment units for our simulation. FAC modules are our Flexible Arithmetic Units, and SFU modules are Special Function Units used to perform special scalar operations like reciprocal square root. We feel that these units are representative of those used in the fragment portion of modern graphics hardware.

NVIDIA's 6800 GT, so we configure Qsilver to model a system with 6 vertex pipelines and 16 fragment pipelines. The fragments are tiled in blocks of 2×2, so we effectively have 4 tile pipelines, each of which can process 4 fragments simultaneously. NV4x GPUs use a similar tiled configuration in the fragment engine.

For these experiments, we enhanced Qsilver to track fragment shader activity. Our modified Qsilver simulator stores a per-triangle identifier which uniquely specifies which, if any, fragment shader was bound when that triangle was being rendered. We also store the text of the fragment shaders so that they can be analyzed by the Qsilver simulator core.

4 Experiments and Results

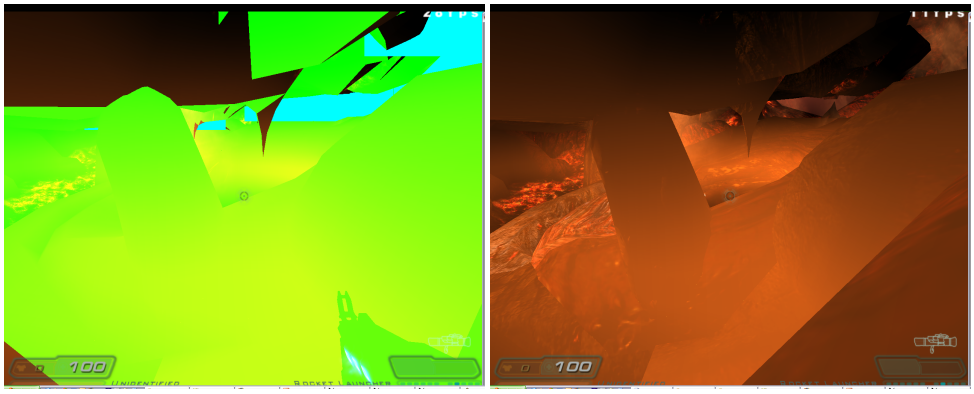
In this section, we describe two experiments we performed to validate our hypothesis that using SSR components in a modern GPU architecture can benefit certain applications. We show improved performance in the recent game Doom III with only a minimal impact on GPU die area and also demonstrate that double-precision floating point capabilities can be added to the fragment pipeline without affecting the performance of single-precision applications.

4.1 Increased Throughput

We first compared the simulated performance of an NV4x-like fragment pipeline to that of an SSR pipeline architecture, whose fragment units are depicted in Figure 2. The fragment units in our target SSR architecture are similar to those found in NV4x GPUs[1]. However, we replace both the multipliers and adders in stages 1 and 2 with single-precision Flexible Arithmetic Units (FACs). FACs can be very quickly reconfigured to perform either a multiplication or an addition and use only slightly more gates than a multiplier. With current technology, these FACs can produce a result every cycle and can be reconfigured between cycles, assuming a 400 MHz clock and a two stage pipeline. This modification gives us significantly more scheduling opportunities, as we can execute three vector additions in a single pass, while an NV4x fragment unit can only perform one [9]. Moreover, there is more freedom to schedule dot product and multiply-accumulate operations.

Given these additional scheduling opportunities and the known scheduling constraints of NV4x GPUs, we hand scheduled fragment programs intercepted from a 50-frame

[1] Using those details that have been made available to the public or indirectly obtained via patents and extensive benchmark tests.



(a) (b)

Figure 3. Screen captures from Doom III. On the left, the color of each pixel is modulated to to indicate which fragment program generated it. The right image is the unmodified rendering from the game. Notice that the majority of pixels are generated by programmable fragment shaders.

Doom III demo (see Figure 3), which was then simulated under Qsilver. An NV4x has dedicated hardware for performing common half-precision operations in parallel with full precision operations; due to limitations with the current NVIDIA drivers, we were forced to require *NVShaderPerf* (a utility that displays shader scheduling information for NVIDIA hardware) to schedule programs for our NV4x-like architecture using the full precision path only. From the simulation of this data stream, we obtained a 4.27% speedup over the entire graphics pipeline for the SSR architecture.

Equally as important, based on conservative gate count estimates, each FAC requires 12,338 gates, only 710 more than a single-precision multiplier (11,628 gates). Replacing the adders (7,782 gates) requires 4,556 additional gates. This additionally requires the small overhead of a multiplexer to configure the FACs. Based on our gate estimates, with 16 fragment pipelines, the cost of our proposed use of SSR is 382,464 gates, which is approximately 0.2% of the total area of NVIDIA's 6800 GT.

4.2 Dual-Mode IEEE Adders and Multipliers

The GPGPU and scientific computing communities would like to have the ability to perform double-precision calculations on the GPU. Unfortunately for them, the gaming industry drives the graphics hardware industry, and games do not require double-precision. We present a method by which both gaming and scientific communities can get what they want.

A dual-mode floating point unit is a small-scale reconfigurable unit capable of performing two simultaneous single-precision operations or one double-precision operation. Dual-mode units can be fully pipelined to produce results every cycle. Like other SSR units, dual-mode multipliers and adders require internal multiplexers for path selection. Additionally, they require a rounding unit capable of flexible rounding modes. The total additional structure for this modification is insignificant. We have conservative gate count estimates for a dual precision FAC of 38,475 gates; this is significantly less than the combined gate count of a dual precision adder (13,456 gates) and multiplier (37,056 gates)—50,512 gates total. Based on component similarities, a configurable single-precision/double-precision FAC, which we describe here, would require approximately the same gate count.

We simulate a pipeline in Qsilver that uses dual-mode multipliers and adders in the fragment engine. Because we have modified only the multiplication and addition units, additional precision is not available for specialized operations such as logarithms or square roots. Although many scientific applications would benefit greatly from high precision addition and multiplication alone, a full double-precision arithmetic engine would be ideal. Dual-mode reciprocal, square-root, logarithm, and other specialized units are a topic for future exploration.

To validate an SSR-based graphics architecture capable of both single- and double-precision, we traced four Brook demo programs through Qsilver:

1. **bitonic_sort**, a parallel sorting network
2. **image_proc(25,25)**, an image convolution shader
3. **particle_cloth(5,10,15)**, a cloth simulation
4. **volume_division(100)**, a volume isosurface extracter.

Demo	**bitonic_sort**	**image_proc**	**particle_cloth**	**volume_division**
32-bit cycles	468	1,292	19,504	254,923,418
64-bit cycles	877	2,525	38,959	509,846,783
32-bit→64-bit speedup	.534	.512	.501	.500

Table 1. Single- and double-precision GPGPU computations using SSR. Each application comes with the Brook distribution. The *32-bit cycles* row shows the GPU cycle count for our NV4x-like architecture. Note that these timings are identical whether we are using a dual-mode unit configured in single-precision mode or a dedicated single-precision unit. The *64-bit cycles* row shows the cycles required for double-precision after reconfiguration. As expected, none of the programs takes more than twice as long with double-precision than with single-precision.

The results are summarized in Table 1. This table lists the cycle counts for each application in both single- and double-precision modes. Because we retask two single-precision FPUs for each double-precision FPU, double-precision calculations effectively cut the throughput of the architecture. Of course, the double-precision calculations never require more than twice as long as the corresponding single-precision calculation. Because the timing results are identical for dual-mode units configured in single-precision mode and dedicated single-precision units, we have shown that by using SSR we can add double-precision addition and multiplication to the graphics pipeline with only a modest increase in gate count and without affecting the performance of the commonly-used single-precision path.

5 Conclusions

We have extended Qsilver to record information on fragment program state in its annotated trace. Our modified Qsilver core then uses this new information, along with fragment program listings and timing information, to model the programmable fragment engine of an NV40-like architecture. With this framework in place, we have demonstrated the applicability of Small-Scale Reconfigurability to graphics architectures. We have shown that it is possible to increase the throughput of the fragment engine with only a small increase in die area. In addition, we have demonstrated that dual-mode multipliers can provide double-precision in the fragment engine to support scientific computing in the GPGPU community with no detriment to the gamers who drive the market. The vector-like operations performed on GPUs make them a particularly good target for such techniques, since need for reconfiguration is rare in SIMD environments, and since the cost of reconfiguration is amortized over many operations.

6 Future Work

The fragment engine represents only a small portion of the graphics pipeline. Applications of SSR will likely yield similar performance improvements in other units as well.

Another area of exploration that is likely to be fruitful for SSR is power consumption. Whenever portions of a chip are unused, they use no dynamic power, but they leak static power. By their very nature, SSR components are rarely idle, and should therefore leak a minimum of static power. Power leakage is currently a major issue with GPUs, and reducing leakage becomes crucial as continuing improvements in chip manufacturing technology exacerbate this problem.

7 Acknowledgments

We would like to thank John Lach for his input on SSR and Peter Djeu for his collaboration on Chromium extensions. This work was funded by NSF grants CCF-0429765, CCR-0306404, and CCF-0205324.

References

[1] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, , and P. Hanrahan. Brook for GPUs: Stream computing on graphics hardware. *ACM Transactions on Graphics*, 2004.

[2] L.-Y. Chiou, S. Bhunia, and K. Roy. Synthesis of application-specific highly efficient multi-mode cores for embedded systems. *ACM Transactions on Embedded Computing Systems*, 2005.

[3] S. Chiricescu, M. Schuette, R. Glinton, and H. Schmit. Morphable multipliers. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2002.

[4] K. Compton and S. Hauck. Flexibility measurement of domain-specific reconfigurable hardware. In *Proceedings of the ACM/SIGDA Symposium on Field-programmable Gate Arrays*, 2004.

[5] G. Even, S. M. Mueller, and P.-M. Seidel. A dual mode IEEE multiplier. In *Proceedings of the International Conference on Innovative Systems in Silicon*, 1997.

[6] L. M. Guerra, M. Potkonjak, and J. M. Rabaey. Behavioral-level synthesis of heterogeneous bisr reconfigurable asic's. *IEEE Transactions on VLSI*, 1998.

[7] G. Humphreys, M. Houston, R. Ng, S. Ahern, R. Frank, P. Kirchner, and J. T. Klosowski. Chromium: A stream processing framework for interactive graphics on clusters of workstations. *ACM Transactions on Graphics*, 21(3):693–702, July 2002.

[8] K. Kim, R. Karri, and M. Potkonjak. Synthesis of application specific programmable processors. In *Proceedings of Design Automation*, 1997.

[9] A. Seifert. NV40 technology explained. `http://3dcenter.org/artikel/nv40_pipeline/index3_e.php`.

[10] J. W. Sheaffer, D. P. Luebke, and K. Skadron. A flexible simulation framework for graphics architectures. In *Proceedings of Graphics Hardware 2004*, Aug. 2004.

[11] J. W. Sheaffer, K. Skadron, and D. P. Luebke. Studying thermal management for graphics-processor architectures. In *Proceedings of 2005 IEEE International Symposium on Performance Analysis of Systems and Software*, Mar. 2005.

[12] V. Vijay Kumar and J. Lach. Designing, scheduling, and allocating flexible arithmetic components. In *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2003.