A Hardware Redundancy and Recovery Mechanism for Reliable Scientific Computation on Graphics Processors

Jeremy W. Sheaffer[1]

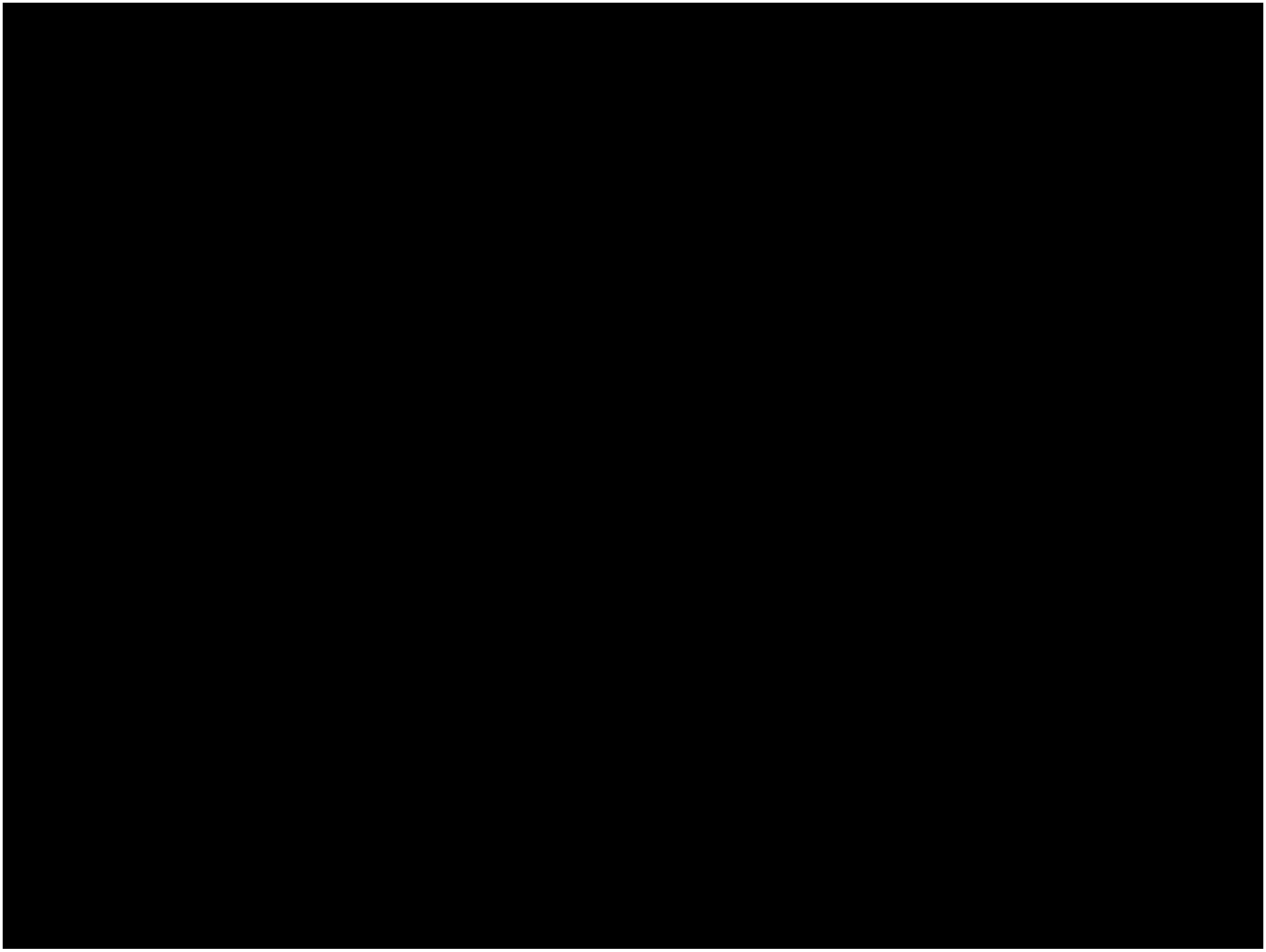
David P. Luebke[2]

Kevin Skadron[1]

[1]University of Virginia Computer Science
[2]NVIDIA Research

Reliable Graphics?

- Transient errors can cause undesirable visual artifacts, such as:
  - Single pixel errors
  - Single texel errors
  - Single vertex errors
  - Corrupt a frame
  - Crash the computer
  - Corrupt rendering state



Motivation

- GPGPU
  - One or more *correct answers* are expected
  - Very different expectation from that of graphics
    - More (exactly) like CPU expectations
  - Massive parallelism provides opportunities that are impractical or impossible on CPUs

Reducing Error Rates

- Error rates can be reduced by
  - Reducing chip operating temperature
  - Reducing crosstalk
  - Increasing transistor sizes
  - Increasing supply voltage
  - Decreasing clock frequency
  - Increasing power supply quality
  - …

Reducing Error Rates

- Error rates can be reduced by
  - …
  - Detection and correction

CPU Transient Fault Mitigation

- ECC and parity
- Scrubbing
  - Used in conjunction with ECC to reduce 2-bit errors
- Larger or radiation-hardened gates
- Hardware fingerprinting or state dump with rollback
- Redundancy
  - Primarily employed to protect logic
  - Also sometimes used for memory

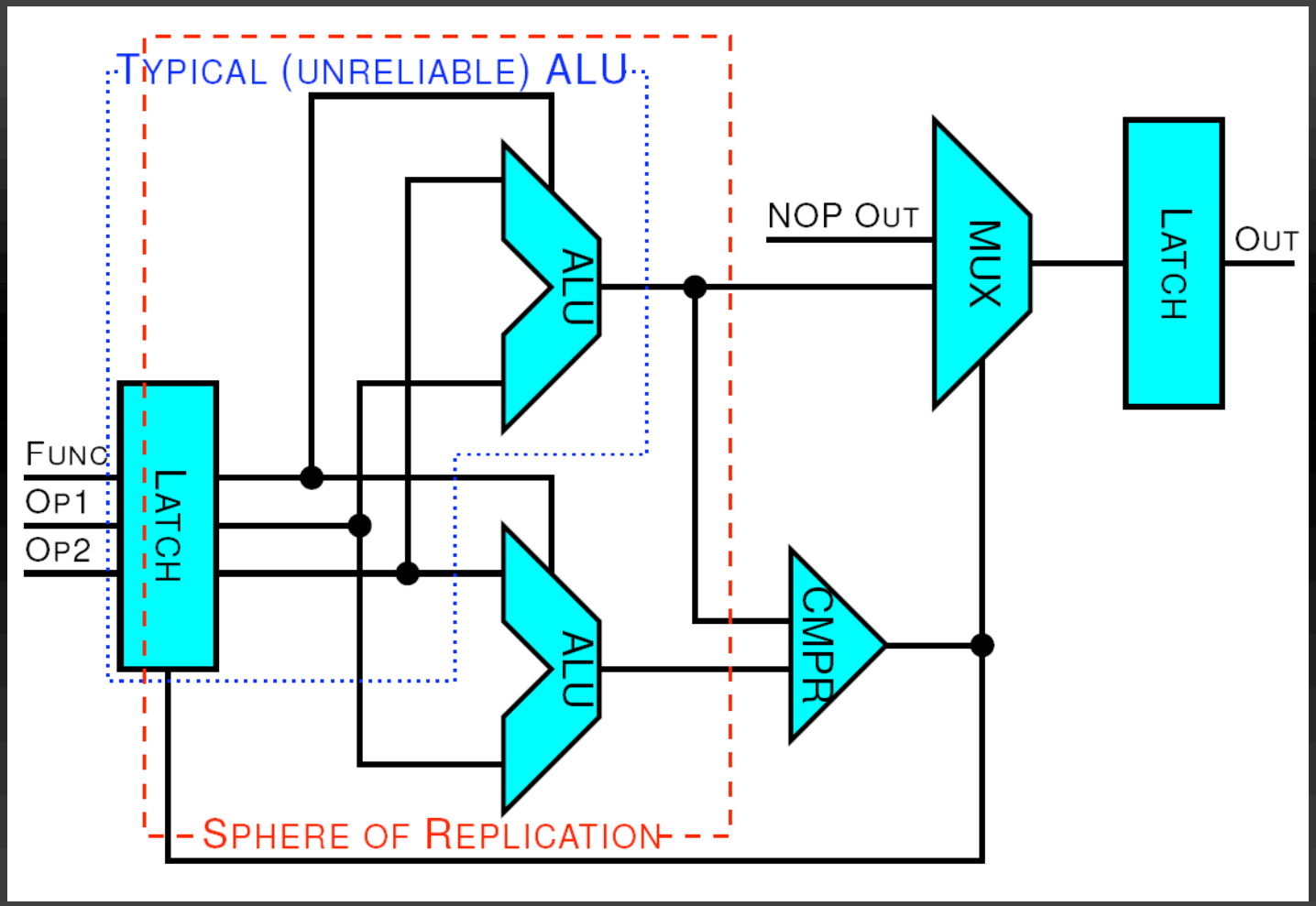
Reliability Through Redundancy

- Primary topic in recent transient fault reliability literature
- Many clever ideas including
  - Triple redundancy with voting
  - Lockstepped processors
  - Redundant Multithreading
    - CRT—Chip-level Redundantly Threaded processors
    - SRT—Simultaneous and Redundantly Threaded processors
      - The concepts of a ‘Sphere of Replication’ and leading and trailing threads
  - Memoization of redundant results

Designing a Reliable Functional Unit

- It is impossible to guarantee 100% reliability
- Anything outside of the *sphere of replication* must be either:
  - Protected, as with ECC, or
  - Unprotected and *unimportant* (as per an ACE analysis)

Example: A Reliable ALU



Motivation for Reliable GPGPU

- GPGPU is becoming important enough that vendors are devoting (human) resources to it
- GPGPU is already being applied in domains where errors are unacceptable
- GPGPU offers a much higher performance per dollar than the traditional supercomputing infrastructure

GPGPU Redundancy

- At what granularity should the redundancy occur?
  - Possibilities include:
    - Multiple GPUs
    - Shader binary (software)
    - Quad/Warp
    - Shader unit (hardware)
    - ALU
  - Tightly coupled with comparator placement and datapath
  - Possible to analytically eliminate many possibilities
    - Experimentally evaluate remaining

Design Concerns

- Solution must not impact graphics performance
- Solution must be very cheap to implement
  - GPU vendors are very reluctant to sacrifice real estate for anything which does not boost performance

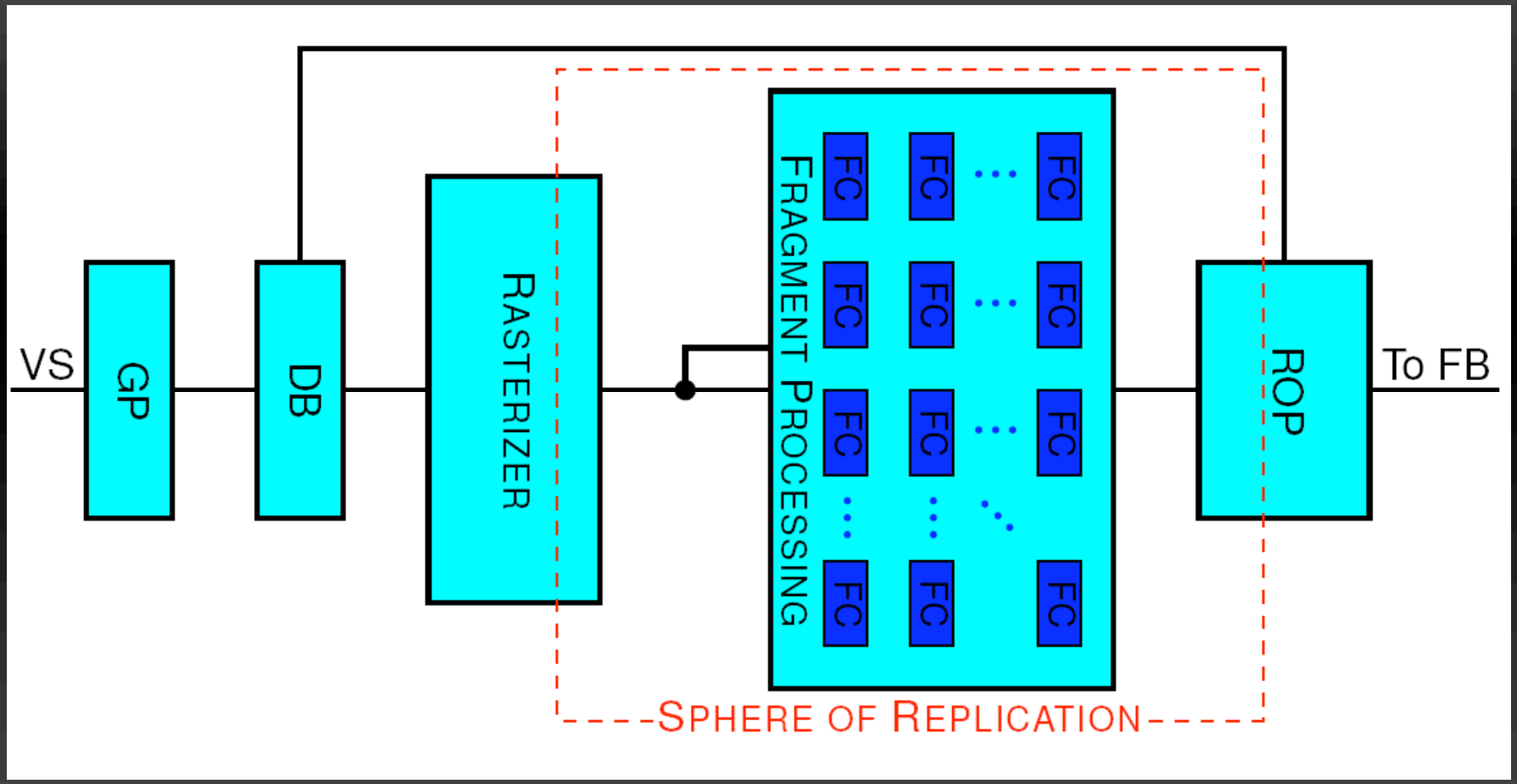
- GPGPU is arguably becoming important
  - But it does not drive the market
    - (and it probably never will)

Performance Concerns

- It should be faster than 2x
- It should use less than 2x energy

- A well designed solution should be able to achieve these goals by taking advantage of increased memory locality of redundant texture fetches

A Reliable GPGPU Solution



A Reliable GPGPU Solution

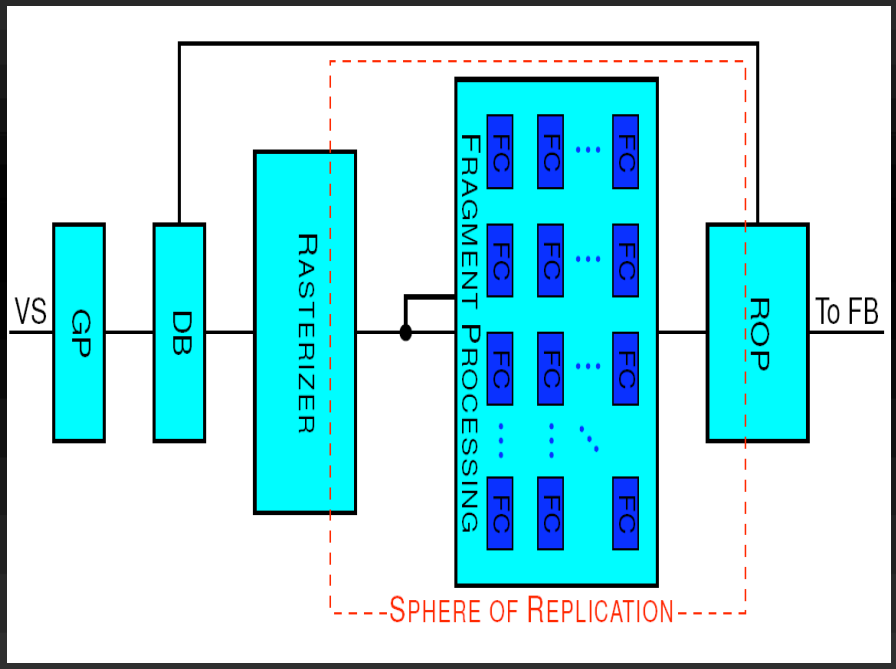
VS: Vertex Stream

DB: *Domain Buffer*

GP: Geometry Processing

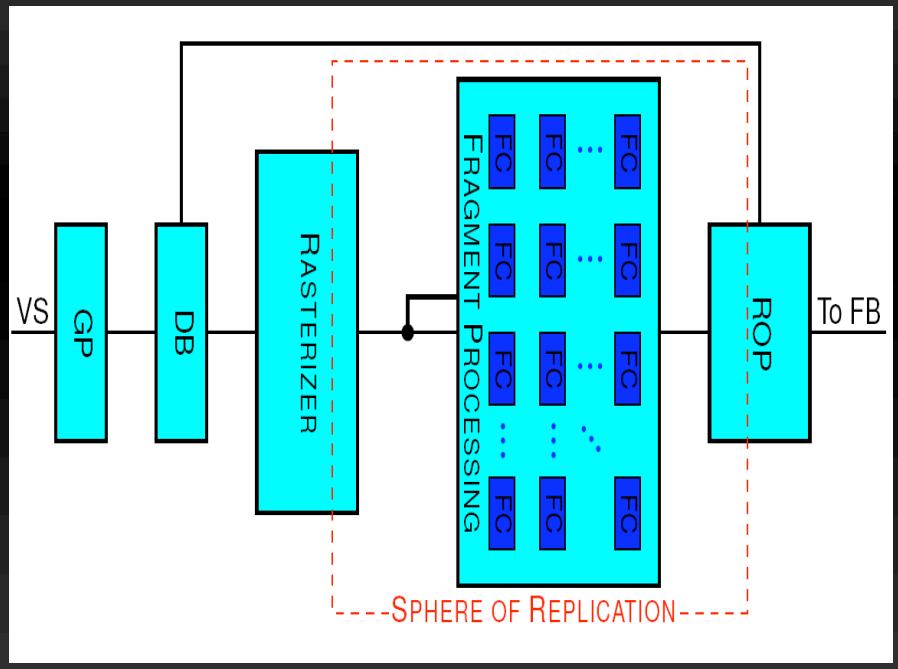
FC: Fragment Core

FB: Framebuffer



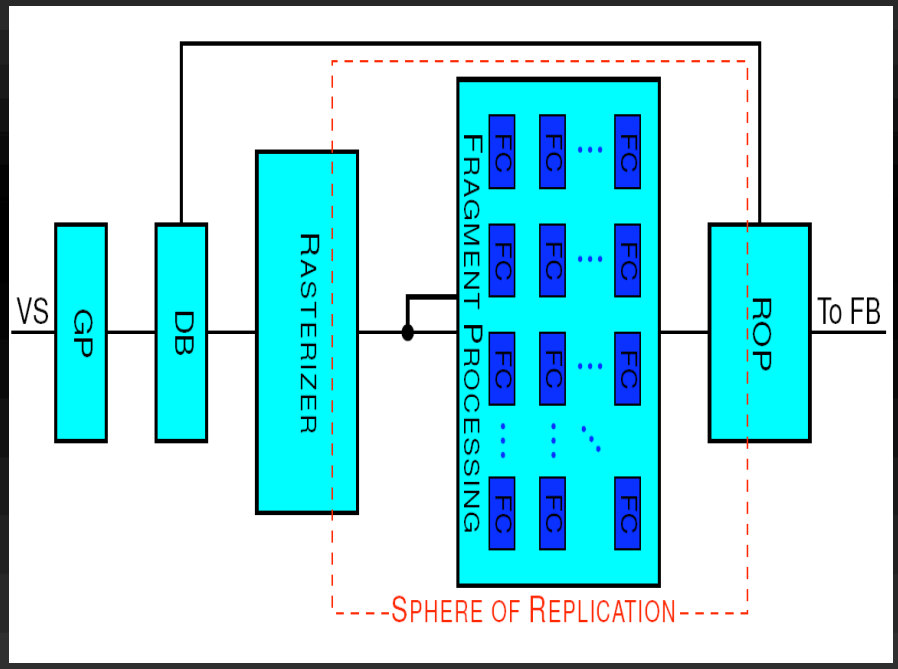
The Domain Buffer

- Stores assembled triangle information in protected memory
- Reads datapath from ROP for reissue
  - Datapath could be repurposed from the unified shader model or f-buffer datapaths
- Reissues with fragment(s) from ROP as stencil mask(s)



Other Pipeline Changes

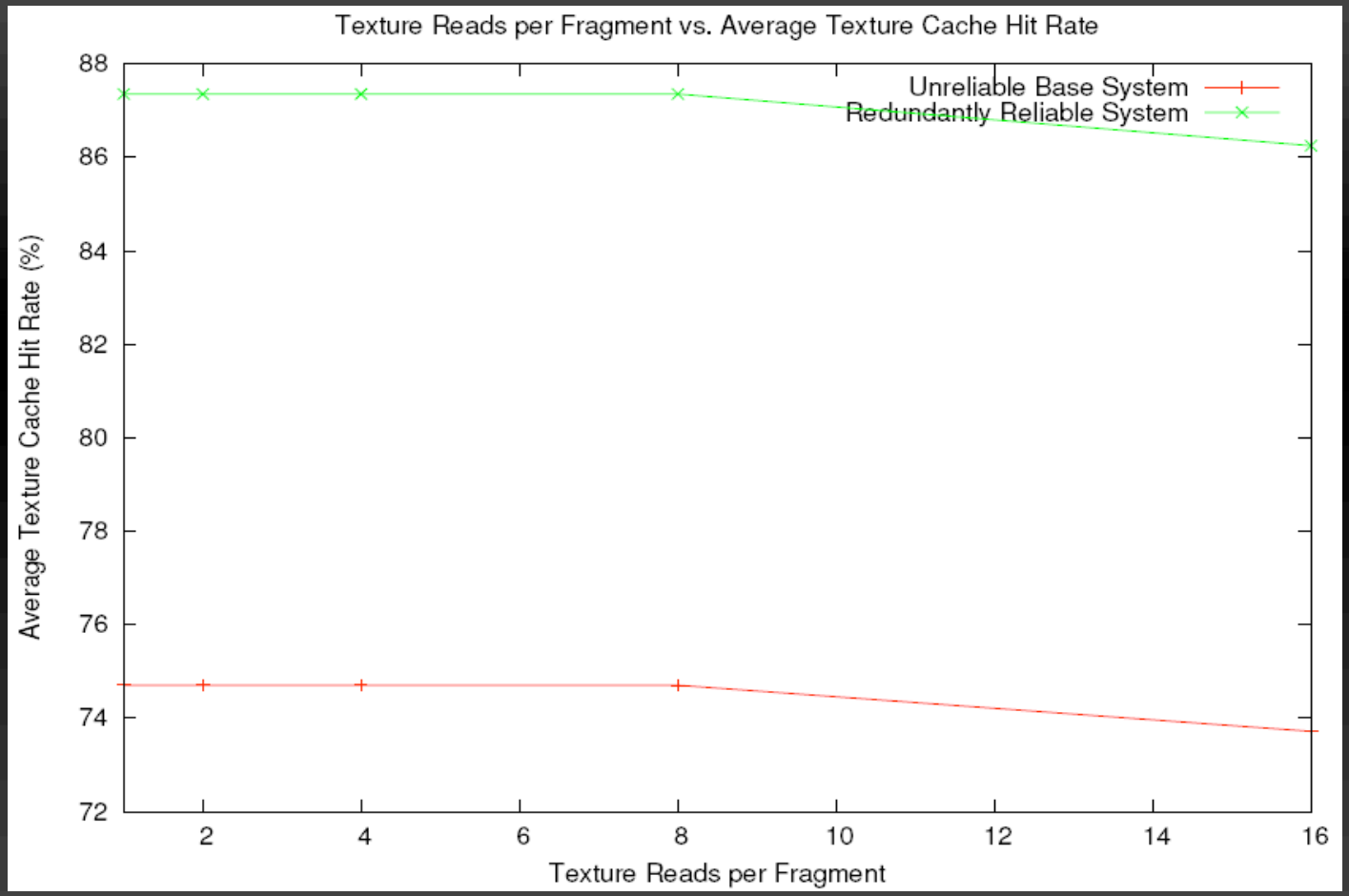
- Rasterizer produces two of every fragment
  - Guarantees that identical fragments are not computed on the same core
- The fragment engine has no changes
- ROP uses a modified full/empty-bit semantic to act as the comparator



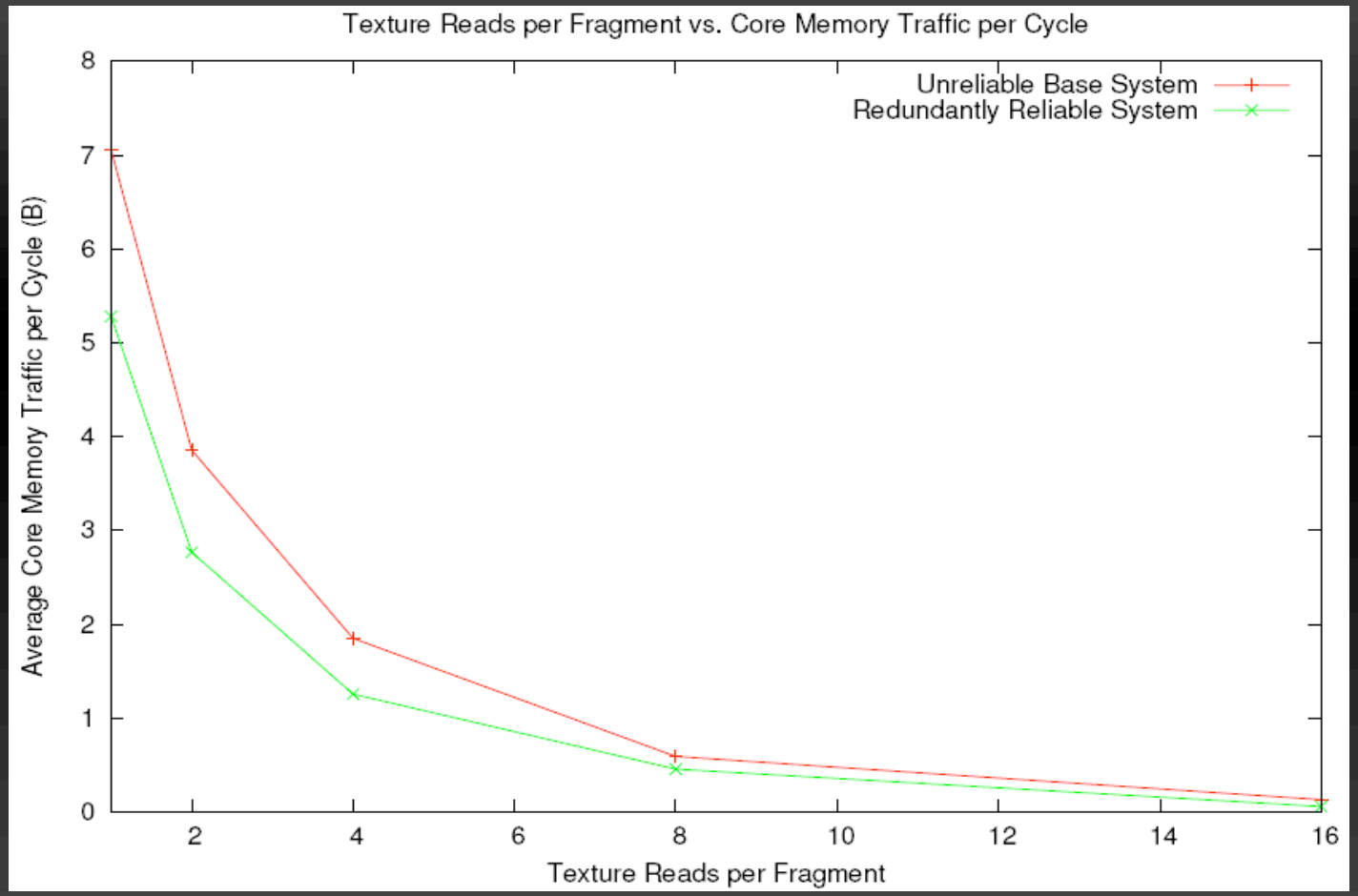
Experiments

- Using a series of stressmarks to challenge the memory system
  - Compare with a baseline, unreliable system and with a *perfect cache* system in which cache accesses never go to memory
    - Measure performance, power, energy, cache hit rate, and memory throughput

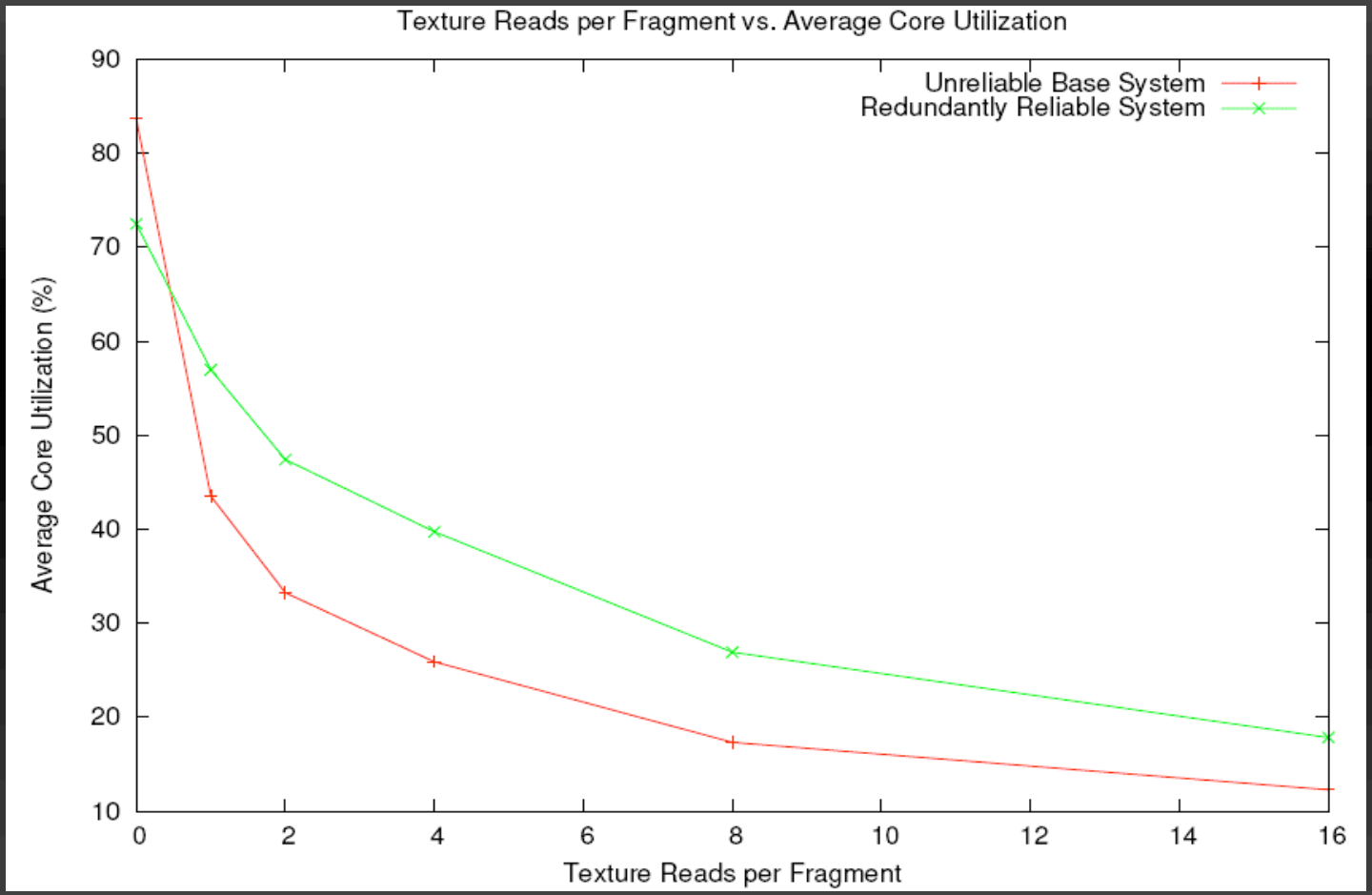
Improved Cache Performance



Reduced Memory Traffic



Better Core Utilization



Less than 2x performance overhead

Memory↓	Domain→	16^2	32^2	64^2	128^2	256^2	512^2
0 reads	Real Cache	1.70	1.38	1.38	1.41	1.30	1.21
	Perfect Cache	1.70	1.38	1.38	1.41	1.30	1.21
1 read	Real Cache	1.72	1.33	1.24	1.22	1.25	1.21
	Perfect Cache	1.72	1.41	1.44	1.43	1.49	1.31
2 reads	Real Cache	1.56	1.32	1.31	1.22	1.25	1.30
	Perfect Cache	1.64	1.47	1.50	1.50	1.62	1.65
4 reads	Real Cache	1.75	1.41	1.30	1.22	1.28	1.31
	Perfect Cache	1.73	1.58	1.57	1.62	1.81	2.13
8 reads	Real Cache	1.49	1.32	1.13	1.21	1.20	N/A
	Perfect Cache	1.78	1.65	1.69	1.71	1.88	2.17
16 reads	Real Cache	1.50	1.22	1.20	1.36	1.57	N/A
	Perfect Cache	1.86	1.74	1.76	1.85	2.02	2.22

Less than 2x power and energy

Memory↓	Domain→	16^2	32^2	64^2	128^2	256^2	512^2
0 reads	Power	1.15	1.40	1.40	1.38	1.50	1.61
	Energy	1.97	1.94	1.94	1.95	1.95	1.95
1 read	Power	1.14	1.44	1.54	1.56	1.54	1.59
	Energy	1.97	1.93	1.92	1.93	1.94	1.94
2 reads	Power	1.24	1.45	1.46	1.56	1.53	1.49
	Energy	1.94	1.92	1.92	1.91	1.93	1.94
4 reads	Power	1.12	1.37	1.47	1.56	1.50	1.47
	Energy	1.96	1.93	1.91	1.91	1.92	1.93
8 reads	Power	1.27	1.43	1.63	1.54	1.56	N/A
	Energy	1.90	1.90	1.85	1.87	1.88	N/A
16 reads	Power	1.27	1.52	1.54	1.37	1.21	N/A
	Energy	1.91	1.87	1.86	1.86	1.90	N/A

Conclusions

- We have presented a reliable GPGPU system
- Our solution utilizes a *domain buffer*, to reissue corrupt fragments, dual issue from the rasterizer, and repurposes ROP as the comparator
- This work provides a complete solution to GPU reliability for last-generation hardware
  - The important ideas map directly to current and foreseeable future hardware, but details become more difficult

Future Work

- Scatter functionality in CTM and CUDA provide difficult challenges
- Other aspects of the presented work map very well to new architectures, though details must be worked out

Acknowledgements

- The simulation framework on which we built this work was developed by Greg Johnson, Chris Burns, Alexander Joly, and William R. Mark at the University of Texas, Austin
- This work was supported by NSF grants CCF-0429765, CCR-0306404, the Army Research Office under grant no. W911NF-04-1-0288, a research grant from Intel MRL, and an ATI graduate fellowship

Thank You

- Questions?