A High-Performance Fault-Tolerant Software Framework for Memory on Commodity GPUs

Anca Grigoras
OUTLINE

• Context
• Fault Tolerant Techniques
• Software Framework
• Performance Results
• Conclusions
• Context
• Fault Tolerant Techniques
• Software Framework
• Performance Results
• Conclusions
Dominant applications of GPUs favor performance over reliability.

Lately, GPUs are increasingly used to accelerate HPC applications.

Many scientific applications require more rigorous error detection and correction (EDAC) capabilities even with extra performance overheads.

Reliability must be reconsidered!!!

• PROBLEM: The current generation of GPUs does not have standard error detection and correction capabilities
• SOLUTION: A software framework that extends standard GPU applications with error detection and recovery with negligible overheads
  • Data coding for error detection
  • Checkpointing for recovery
• DRAM bit-flip errors:
  • are transient
  • can be caused by energetic particles
  • can cause fail-stop program crashes and silent data errors
  • are mostly single-bit errors
  • use parity and Hamming codes as detection and correction techniques
  • double-bit errors are rare
• Context
• Fault Tolerant Techniques
• Software Framework
• Performance Results
• Conclusions
• Information redundancy is attained from:
  • Spatial or temporal redundancy
  • EDAC coding
• Most used approach: hardware-based (72,64) Hamming ECC
  • Simple
  • Single-error correction and double-error detection (SEC-DED)
• Software implementation of SEC-DED ECC in CUDA
  • achieve a high level of fault-tolerance
  • two optimizations => minor overhead

1. Checkpointing
  • large cost of computation to correct the errors
  • alternative: combine error-detection code with checkpoint-based multiple executions
  • allows us to re-execute the kernel
• Software implementation of SEC-DED ECC in CUDA
  • achieve a high level of fault-tolerance
  • two optimizations => minor overhead

1. Checkpointing
  • large cost of computation to correct the errors
  • alternative: combine error-detection code with checkpoint-based multiple executions
  • allows us to re-execute the kernel
2. Blocking

- reduce data coding overheads
- global memory is optimized for block data accesses
- in standard cross-parity code: horizontal and vertical coding
- the modified cross-parity code: replaced horizontal coding with diagonal one
- apply the code to a block of 32 4-byte words
- words of data join coding of cross parity by using shared memory
2. Blocking

- A half of threads stores their words to shared memory.
- The words are xor’d to the ones held by another half of threads.
- Each word is rotated in diagonal coding.
- One word of parity for both vertical and diagonal parity.
• Context
• Fault Tolerant Techniques
• Software Framework
• Performance Results
• Conclusions
SOFTWARE FRAMEWORK

• Fault-tolerance methodology for CUDA
  • bit flips in GPU DRAM
  • protect a large array of data allocated in global memory

1. Input Transfer
  • data is transferred to global memory
  • check bits for input data are generated on the host CPU
  • =>2 sources of performance overheads: data coding on CPU and code transfer to global memory
SOFTWARE FRMework

1. Input Transfer
   • Data coding on CPU
     • Loading the GPU input data from the host DRAM
     • Coding of the data
     • Sending the check bits to GPU
   • The framework
     • hides the coding cost by overlapping it with PCI transfer
     • Includes a library that specifies which GPU data must be checkpointed
2. Kernel execution
   • the framework:
     • checks integrity of data
     • stores check bits for written data
   • CONCERN: the framework has to be manually implemented

3. Output transfer
   • error checking on GPU global memory or
   • check data on GPU after it is transferred to host
• Context
• Fault Tolerant Techniques
• Software Framework
• Performance Results
• Conclusions
PERFORMANCE RESULTS

- N-body performance comparison
PERFORMANCE RESULTS

• Performance comparison of matrix multiplications

![Graph showing performance comparison of matrix multiplications](image)
• Context
• Fault Tolerant Techniques
• Software Framework
• Performance Results
• Conclusions
CONCLUSIONS

- We need to enhance the reliability of GPUs
- A high-performance software framework for DRAM errors in GPUs
- It combines GPU-optimized coding for error detection with checkpoint based error recovery
- The framework adds little overheads to the performance
THE PROTECTION OF DRAM ERRORS IS AN IMPORTANT STEP FOR RELIABLE GPU COMPUTING