



Understanding Software Approaches for GPGPU Reliability

Martin Dimitrov* Mike Mantor† Huiyang Zhou*

*University of Central Florida, Orlando

†AMD, Orlando



Motivation

- Soft-error rates are predicted to grow exponentially in future process generations. Hard errors are gaining importance.
- Current GPUs do not provide hardware support for detecting soft or hard errors.
- Near future GPUs are not likely to address these reliability challenges, because GPU design are still largely driven by video games market.



Our Contributions

- Propose and evaluate three different software-only approaches for providing redundant computations on GPUs
 - R-Naïve
 - R-Scatter
 - R-Thread
- Evaluate the tradeoffs of some additional hardware support (parity protection in memory) to our software approaches



Presentation Outline

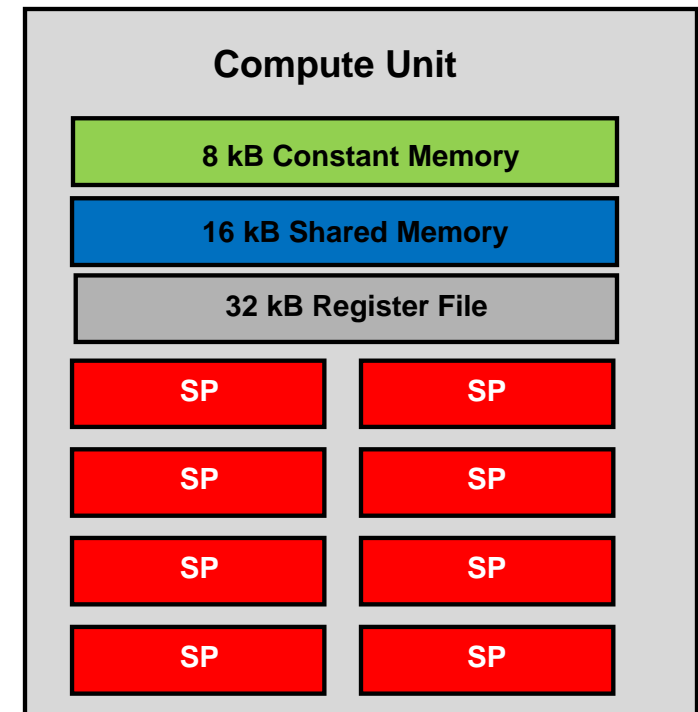
- Motivation
- Our Contributions
- Background on GPU architectures
- Proposed Software Redundancy Approaches
 - R-Naïve
 - R-Scatter
 - R-Thread
- Experimental Methodology
- Experimental Results
- Conclusions



GPU Architectures

NVIDIA G80

- 16 Compute units
 - 8 streaming processors
 - 8K-entry reg. file
 - 16kB shared memory
 - 8kB constant memory
- Huge number of threads can be assigned to a compute unit, up to 512
(8k registers/ 512 threads = 16 registers per thread)
- Threads are scheduled in “warps” of 32

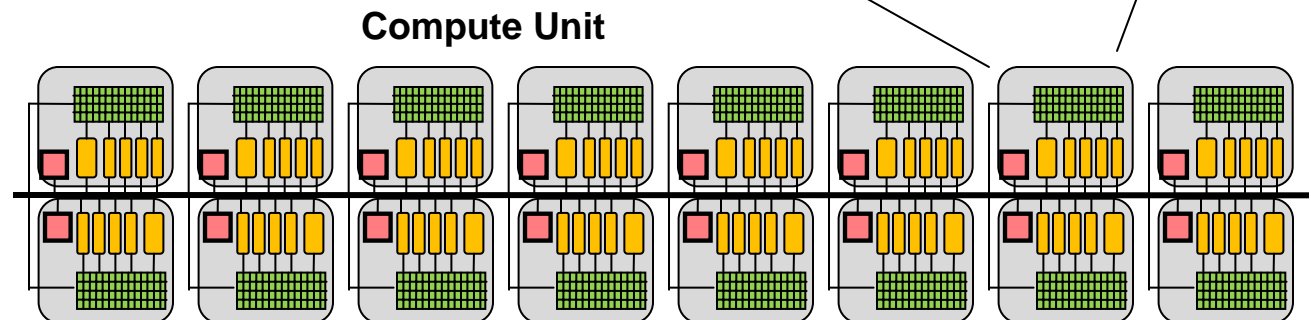
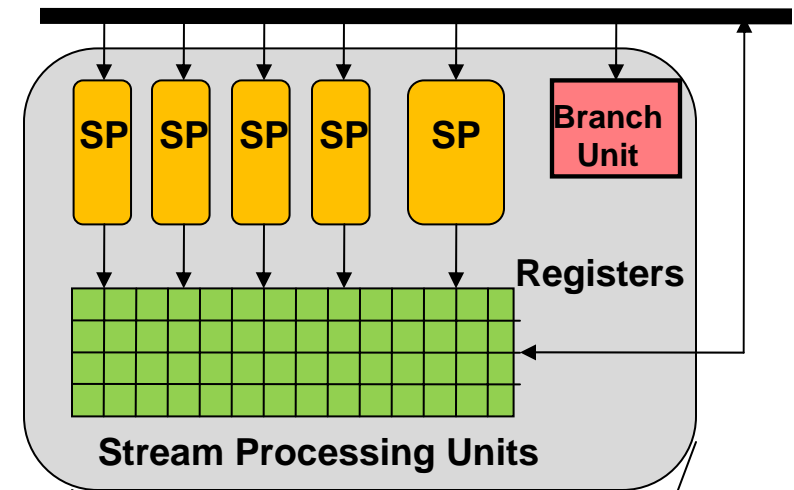




GPU Architectures

ATI R670

- 4 Compute units (CU)
 - 80 streaming processors/CU
 - 256kB register file/CU
- Thread are organized in wavefronts (similar to warps)
- Instructions are grouped into VLIW words of 5





Proposed Software Redundancy Approaches

- Our goal is to provide 100% redundancy on the GPU.
- Duplicate memcopy CPU-GPU-CPU (spatial redundancy for GPU memory).
- Duplicate kernel executions (temporal redundancy for computational logic and communication links)



Proposed Software Redundancy Approaches

R-Naive

StreamRead(in)	StreamRead(in) StreamRead(in_R)	StreamRead(in) Kernel(in,out)	StreamRead(in) Kernel(in,out) StreamWrite(out)
Kernel(in, out)	Kernel(in,out) Kernel(in_R,out_R)	StreamRead(in_R) Kernel(in_R,out_R)	StreamRead(in_R) Kernel(in_R,out_R)
StreamWrite(out)	StreamWrite(out) StreamWrite(out_R)	StreamWrite(out) StreamWrite(out_R)	StreamWrite(out_R)
(a) Original Code	(b) Redundant code without overlap	(c) Redundant code with overlap	(d) Redundant code back-to-back



Proposed Software Redundancy Approaches

R-Naive

- Hard-error: it is desirable for the original and redundant input/output streams to use different communication links and compute cores.
- Solutions
 - For some applications this can be achieved at the application level by rearranging the input data.
 - For other applications it is desirable to have a software controllable interface to assign the hardware resources.



Proposed Software Redundancy Approaches

R-Naive

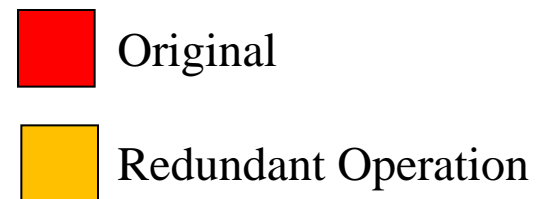
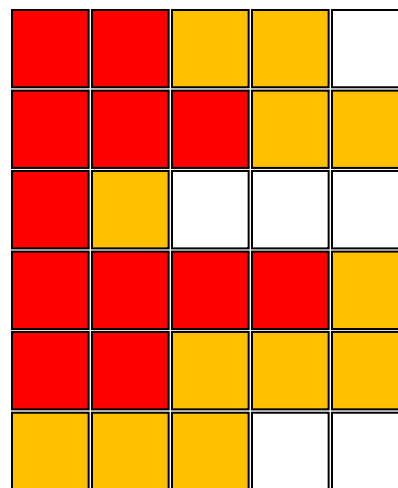
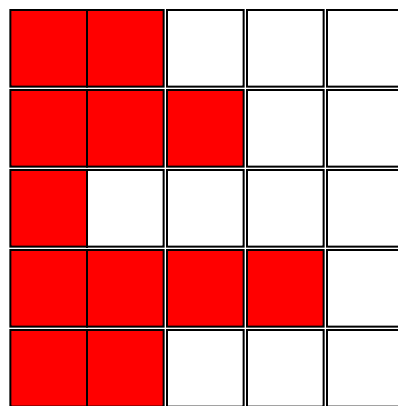
- Advantages/Disadvantages of using R-Naive:
 - + Easy to implement
 - + Predictable performance
 - 100% performance overhead



Proposed Software Redundancy Approaches

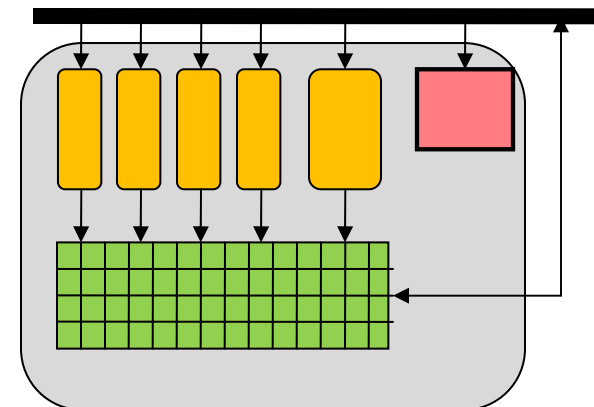
R-Scatter

- Take advantage of unused instruction level-parallelism.



Original vs. R-Scatter VLIW instruction schedules

Each VLIW instruction is mapped to a stream processing unit





Proposed Software Redundancy Approaches

R-Scatter

```
kernel void mat_mult(float width, float M[],
float N[], out float P<>){

float2 vPos = indexof(P).xy; // obtain position into the stream
float4 index = float4(vPos.x, 0.0f, 0.0f, vPos.y);
float4 step = float4(0.0f, 1.0f, 1.0f, 0.0f);
float sum = 0.0f;

for(float i=0; i<width; i=i+1){
    sum += M[index.zw]*N[index.xy]; //accessing input stream
    index += step;
}
P = sum;
}
```

The redundant operations are inherently independent.

input stream

```
kernel void mat_mult(float width, float M[],
float M_R[], float N[], float N_R[],
out float P<>, out float P_R<>){

float2 vPos = indexof(P).xy; // obtain position into the stream
float4 index = float4(vPos.x, 0.0f, 0.0f, vPos.y);
float4 step = float4(0.0f, 1.0f, 1.0f, 0.0f);
float sum = 0.0f;
float sum_R = 0.0f;

for(float i=0; i<width; i=i+1){
    sum += M[index.zw]*N[index.xy]; //accessing input stream
    sum_R += M_R[index.zw]*N_R[index.xy];
    index += step;
}
P = sum;
P_R = sum_R;
}
```

An error to “i” will affect both the original and redundant computation.

(a) Original Code (7 VLIW words)

(b) R-Scatter Code (11 VLIW words)



Proposed Software Redundancy Approaches

R-Scatter

- Advantages/Disadvantages of using R-Scatter:
 - + Better utilized VLIW schedules
 - + Reused instructions (such as the for-loop)
 - + Overlapped memory accesses
 - Extra registers or shared memory used per kernel may affect thread-level parallelism



Proposed Software Redundancy Approaches

R-Thread

- Take advantage of unused thread level-parallelism (unused compute units)
- Allocate double the number of thread-blocks. The extra thread blocks perform redundant computations.

```
float Pvalue = 0;
for (int k = 0; k < Block_Size; ++k){
    float m = M[addr_md];
    float n = N[addr_nd];
    Pvalue += m * n;
}
P[ty*Width + tx] = Pvalue;
```

(a) Original Code

```
if(by >= NumBlocks){
    M = M_R;
    N = N_R;
    P = P_R;
    by = by - NumBlocks;
}
float Pvalue = 0;
for (int k = 0; k < Block_Size; ++k){
    float m = M[addr_md];
    float n = N[addr_nd];
    Pvalue += m * n;
}
P[ty*Width + tx] = Pvalue;
```

(b) R-Thread Code



Proposed Software Redundancy Approaches

R-Thread

- Advantages/Disadvantages of using R-Thread:
 - + Easy to implement
 - + May result in performance improvement if there is not enough thread-level parallelism
 - 100% performance overhead if enough thread-level parallelism is present.



Hardware Support for Error Detection in Off-Chip and On-Chip Memories

- Protecting Off-Chip global memory
 - Benefits all proposed approaches
 - Eliminates the need for a redundant CPU-GPU transfer
- Protecting On-Chip caches, shared memory
 - Benefits R-Scatter on the G80
 - Required for R670 to obtain benefit of protecting off-chip memory due to implicit caching
- Results are compared on the CPU, thus we still need the redundant memory transfer GPU-CPU



Experimental Methodology

Machine Setup

- Brook+ Experiments
 - Brook+ 1.0 Alpha, Windows XP
 - 2.4 GHz Intel Core2 Quad CPU, 3.25 Gbytes of RAM
 - ATI R670 card with 512 MB memory
- CUDA Experiments
 - CUDA SDK 1.1, Fedora 8 Linux
 - 2.3 GHz Quad core Intel Xeon , 2GByte of RAM
 - NVIDIA GTX 8800 card with 768 MB memory
- Both machines have PCIe x16 to provide 3.2 GB/s bandwidth between GPU and CPU.



Experimental Methodology

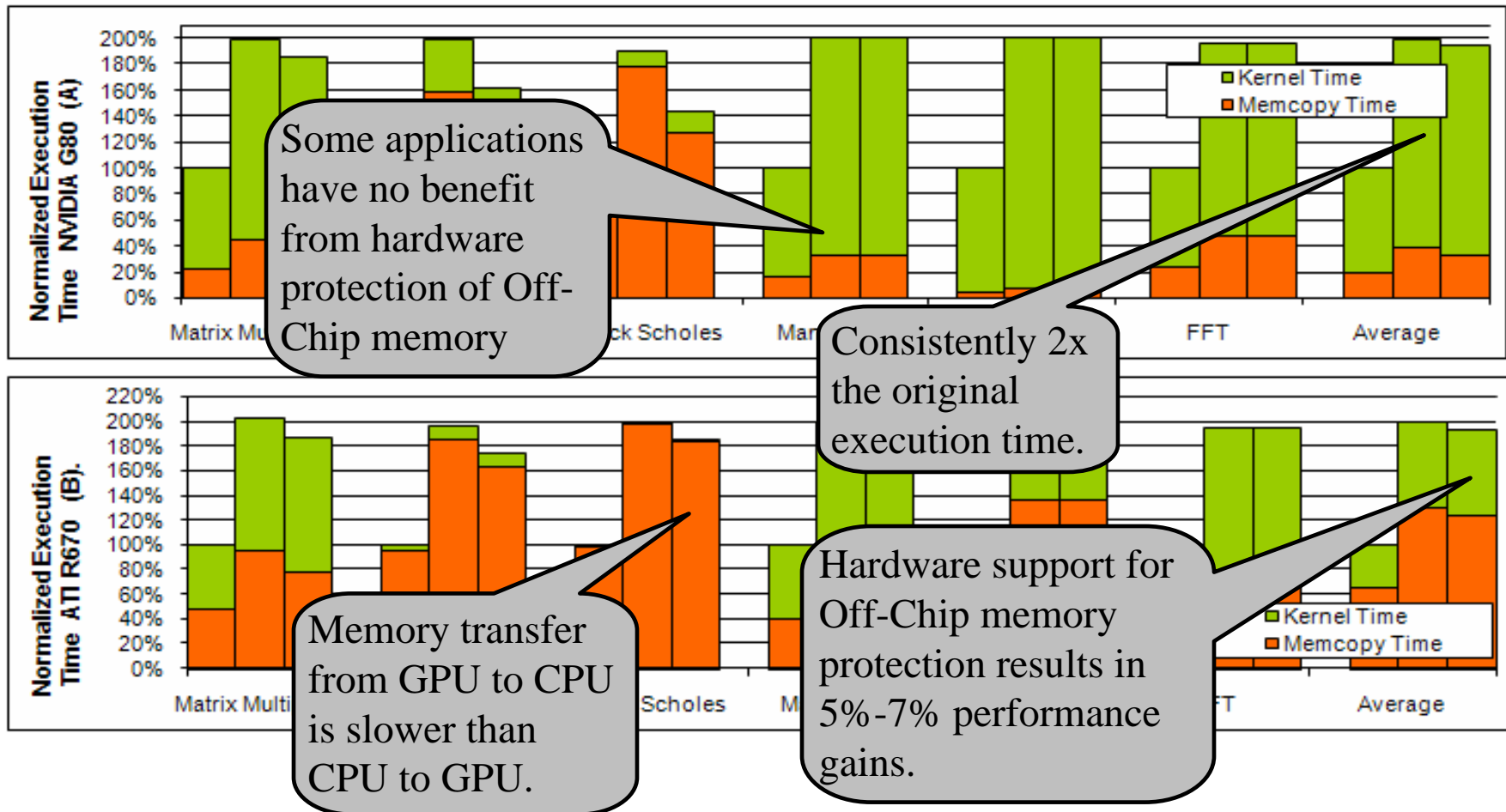
Evaluated Applications

Benchmark Name	Description	Application Domain
Matrix Multiplication	Multiplying two 2k by 2k matrices	Mathematics
Convolution	Applying a 5x5 filter on a 2k by 2k image	Graphics
Black Scholes	Compute the pricing of 8 million stock options	Finance
Mandelbrot	Obtain a Mandelbrot set from a quadratic recurrence equation	Mathematics
Bitonic Sort	A parallel sorting algorithm. Sort 2^{20} elements	Computer Science
1D FFT	Fast Furrier Transform on a 4K array	Mathematics



Experimental Results

R-Naive

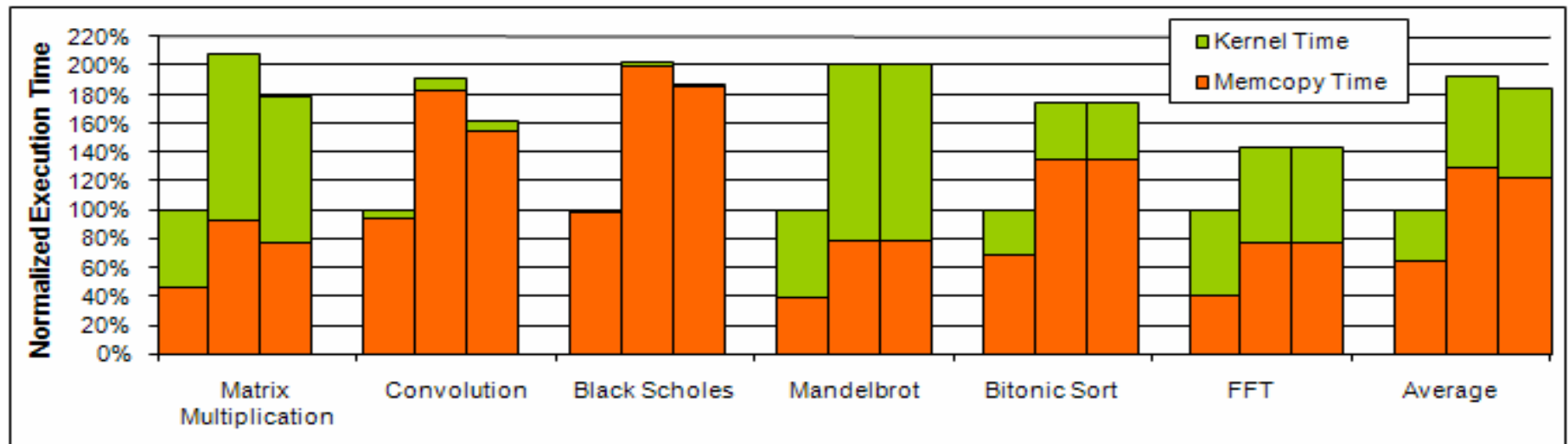




Experimental Results

R-Scatter on R670

- Applications with compacted schedules generally see benefit from R-Scatter (FFT, Bitonic Sort)
- Some applications are still dominated by memory transfer time (Convolution, Black Scholes)
- On average R-Scatter is 195% of the original execution time (185% with hardware memory protection)

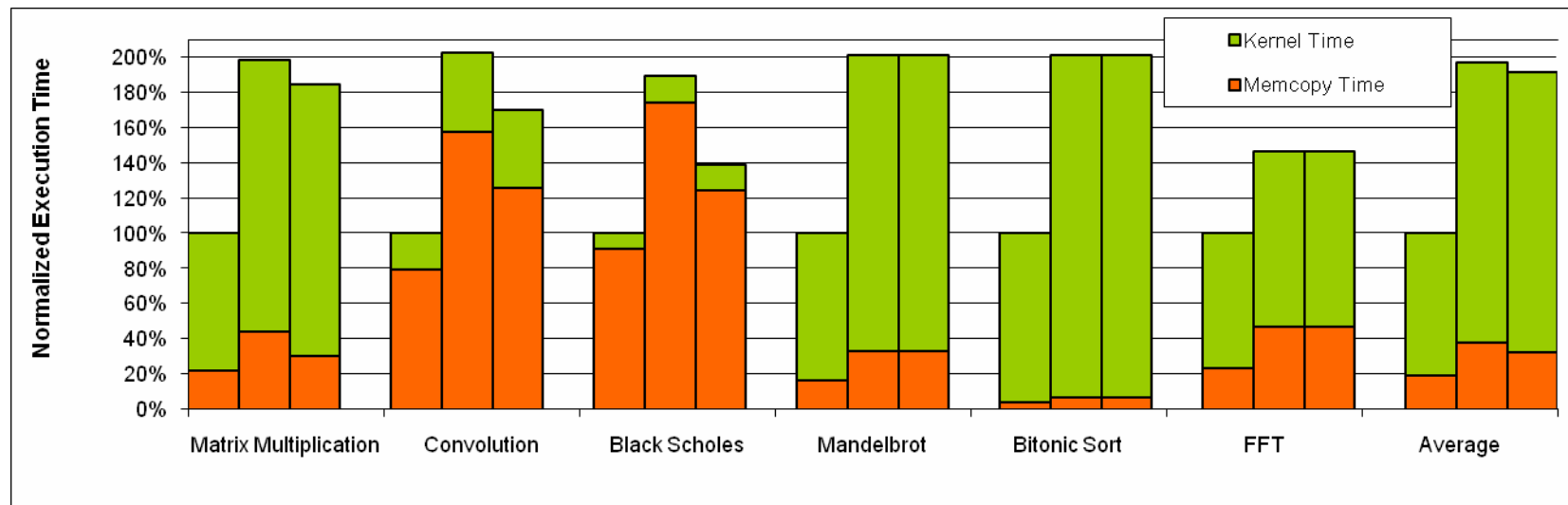




Experimental Results

R-Thread on G80

- Performance overhead uniformly close to 100% due to enough thread-level parallelism our benchmarks.
- When the input size is reduced (exposing some thread-level parallelism) there are clear benefits.





Conclusions

- We proposed three software redundancy approaches with different trade-offs.
- Compiler analysis should be able to utilize some of the unused resources and provide reliability automatically
- We conclude that for our current software approaches, hardware support provides very limited benefit.



Questions





Experimental Results

R-Scatter

VLIW Instruction Schedules for Bitonic Sort on R670.
Each word may contain up to 5 instructions – x,y,z,w,t.

```

16 x: MUL_e      _____, T1.w, T3.z
      y: FLOOR    _____, T0.z
      z: SETGE    _____, T0.y, |KC0[5].x|

17 x: CNDE      T1.x, PV16.z, T0.y, T0.w
      y: FLOOR    T1.y, PV16.x
      z: ADD      T0.z, PV16.y, 0.0f

18 x: MOV       T0.x, |PV17.y|
      y: ADD      _____, |KC0[5].x|, PV17.x
      w: MOV/2    _____, |PV17.y|

19 z: TRUNC     _____, PV18.w
      w: CNDGT    _____, -T1.x, PV18
  
```

(a) Original Code

```

16 x: SETGE     _____, PS15, |KC0[5].x|
      y: ADD      _____, T1.w, KC0[2].x
      z: MULADD   T2.z, -T0.y, T2.x, T1.x
      w: ADD      _____, -|KC0[5].x|, PS15
      t: ADD      _____, T1.w, KC0[8].x

17 x: ADD       _____, -|KC0[11].x|, PV16.z
      y: SETGE    _____, PV16.z, |KC0[11].x|
      z: CNDE     T3.z, PV16.x, T1.y, PV16.w
      w: FLOOR    _____, PV16.y
      t: FLOOR    _____, PS16

18 x: ADD       R2.x, PV17.w, 0.0f
      y: ADD      _____, |KC0[5].x|, PV17.z
      z: ADD      R1.z, PS17, 0.0f
      w: CNDE     T0.w, PV17.y, T2.z, PV17.x
      t: MUL_e    _____, T1.w, T2.y

19 x: FLOOR    R0.x, PS18
      y: MUL_e    _____, T1.w, T3.x
      z: ADD      _____, |KC0[11].x|, PV18.w
      w: CNDGT    _____, -T3.z, PV18.y, T3.z
  
```

(b) R-Scatter Code



Experimental Results

R-Scatter

Benchmark Name	Original VLIW	Original TEX	Original GPR	R-Scatter VLIW	R-Scatter TEX	R-Scatter GPR
Matrix Multiplication	33	8	15	64	16	32
Convolution	21	3	8	29	6	12
Black Scholes	66	5	7	111	10	12
Mandelbrot	19	0	9	33	0	15
Bitonic Sort	39	2	4	46	4	5
1D FFT	72	4	6	78	8	8



Experimental Results

R-Scatter

Benchmark Name	Original VLIW	Original TEX	Original GPR	R-Scatter VLIW	R-Scatter TEX	R-Scatter GPR
Bitonic Sort	39	2	4	46	4	5
1D FFT	72	4	6	78	8	8

- Some applications result in significantly better packed VLIW schedules



Experimental Results

R-Scatter

Benchmark Name	Original VLIW	Original TEX	Original GPR	R-Scatter VLIW	R-Scatter TEX	R-Scatter GPR
Matrix Multiplication	33	8	15	64	16	32

- Matrix multiplication has a compact schedule to begin with. No room for improvement.



Experimental Results R-Scatter

Benchmark Name	Original VLIW	Original TEX	Original GPR	R-Scatter VLIW	R-Scatter TEX	R-Scatter GPR
Mandelbrot	19	0	9	33	0	15

- Too many branch instructions prevent Mandelbrot from creating more compact schedules.



GPGPU Programming Models

CUDA vs. Brook+

- Thread management model CUDA
 - Threads are created explicitly and arranged into a thread hierarchy.
 - Threads => 3D thread blocks => 2D grid
 - Threads query their thread and block ID at runtime

```
__global__ void  
scale (float k, float* a, float *b)  
{  
    int x = threadIdx.x;  
    b[x] = a[x] * k;  
}
```



GPGPU Programming Models

CUDA vs. Brook+

- Thread management model Brook+ 1.0
 - Threads are created implicitly
 - An implicit thread is created for each streaming element
 - Number of threads created depends on the size of the data stream
 - Scatter/Gather streams are also available with some restrictions

```
kernel void
scale (float k, float a<>, out float b<>)
{
    b = a * k;
}
```