Massive Threading: Using GPUs to Increase the Performance of Digital Forensics Tools

By

Lodovico Marziale, Golden Richard and Vassil Roussev

Presented At

The Digital Forensic Research Conference

DFRWS 2007 USA Pittsburgh, PA (Aug 13th - 15th)

Massive Threading: Using GPUs to Increase the Performance of Digital Forensics Tools

Lodovico Marziale, **Golden G. Richard III,** Vassil Roussev

THE UNIVERSITY of NEW ORLEANS

Me:

Professor of Computer Science

Co-founder, Digital Forensics Solutions

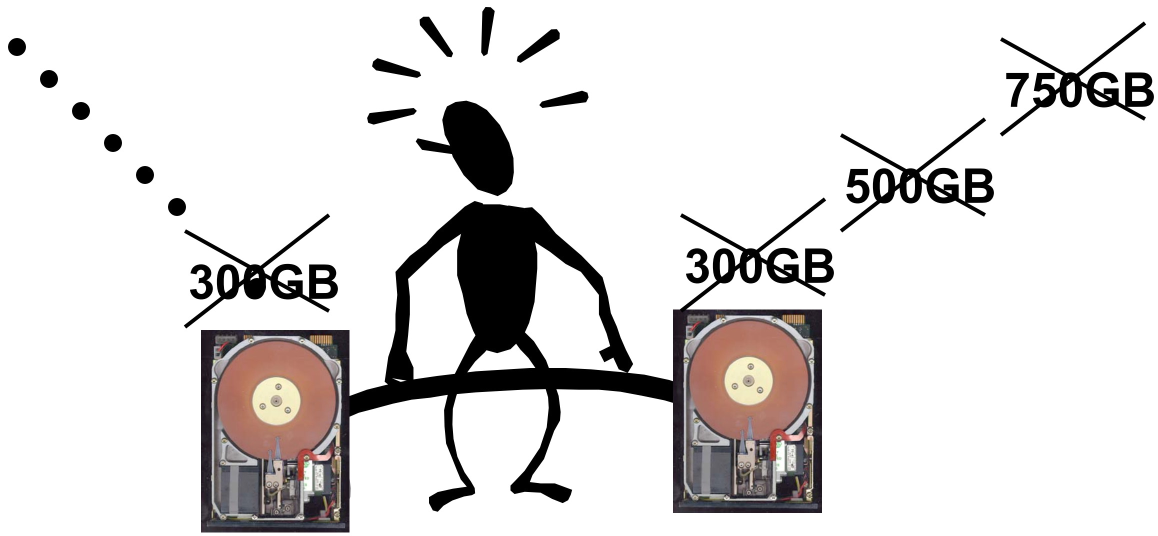
golden@cs.uno.edu

golden@digitalforensicssolutions.com

golden@digdeeply.com

THE UNIVERSITY of NEW ORLEANS

Problem: (Very) Large Targets



- Slow Case Turnaround
- Need:
  - Better software designs
  - More processing power
  - Better forensic techniques

Finding More Processing Power

Filling this gap?

Graphics Processing Units (GPUs)?

speed

speed

speed

Single CPU **Multicore CPUs** **Clusters**

Quick Scalpel Overview

- Fast, open source file carver
- Simple, two-pass design
- Supports "in-place" file carving
- "Next-generation" file carving will use a different model
  - Headers/footers/other static milestones are "guards"
  - Per-file type code performs deep(-er-er?") analysis to find file / fragment boundaries and do reassembly
- But that's not the point of the current work
- Use Scalpel as a laboratory for investigating the use of GPUs in digital forensics
- First, multicore discussion

Multicore Support for Scalpel

- Parallelize first pass over image file
- Thread pool: Spawn one thread for each carving rule
- Loop
  - Threads sleep
  - Read 10MB block of disk image
  - Threads wake
  - Search for headers in parallel
    - Boyer-Moore binary string search (efficient, fast)
  - Threads synchronize then sleep
  - Selectively search for footers (based on discovered headers)
  - Threads wake
- End Loop
- Simple multithreading model yields **~1.4 – 1.7 X** speedup for large, in-place carving jobs on multicore boxes

first pass over image file

Hard to find forensics software that **doesn't** need to do binary string searches

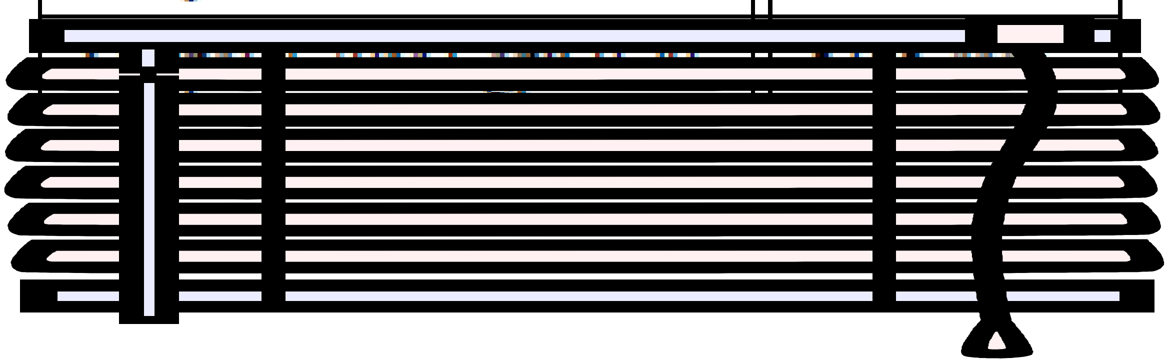
Multicore (2)

TABLE II

RESULTS FOR CARVING 100GB DISK IMAGE ON DUAL PROCESSOR, DUAL CORE SUN ULTRA 40 (2.6GHZ AMD OPTERON 2218 PROCESSORS, 16GB RAM). 30 FILE TYPES, ~15M FILES CARVED. EACH RESULT IS THE AVERAGE OF MULTIPLE, SEQUENTIAL RUNS.

Scalpel 1.60 “vanilla”	13067 secs
Scalpel 1.60 “new q”	8725 secs
Scalpel 1.70MT-multicore	4958 secs

blinds

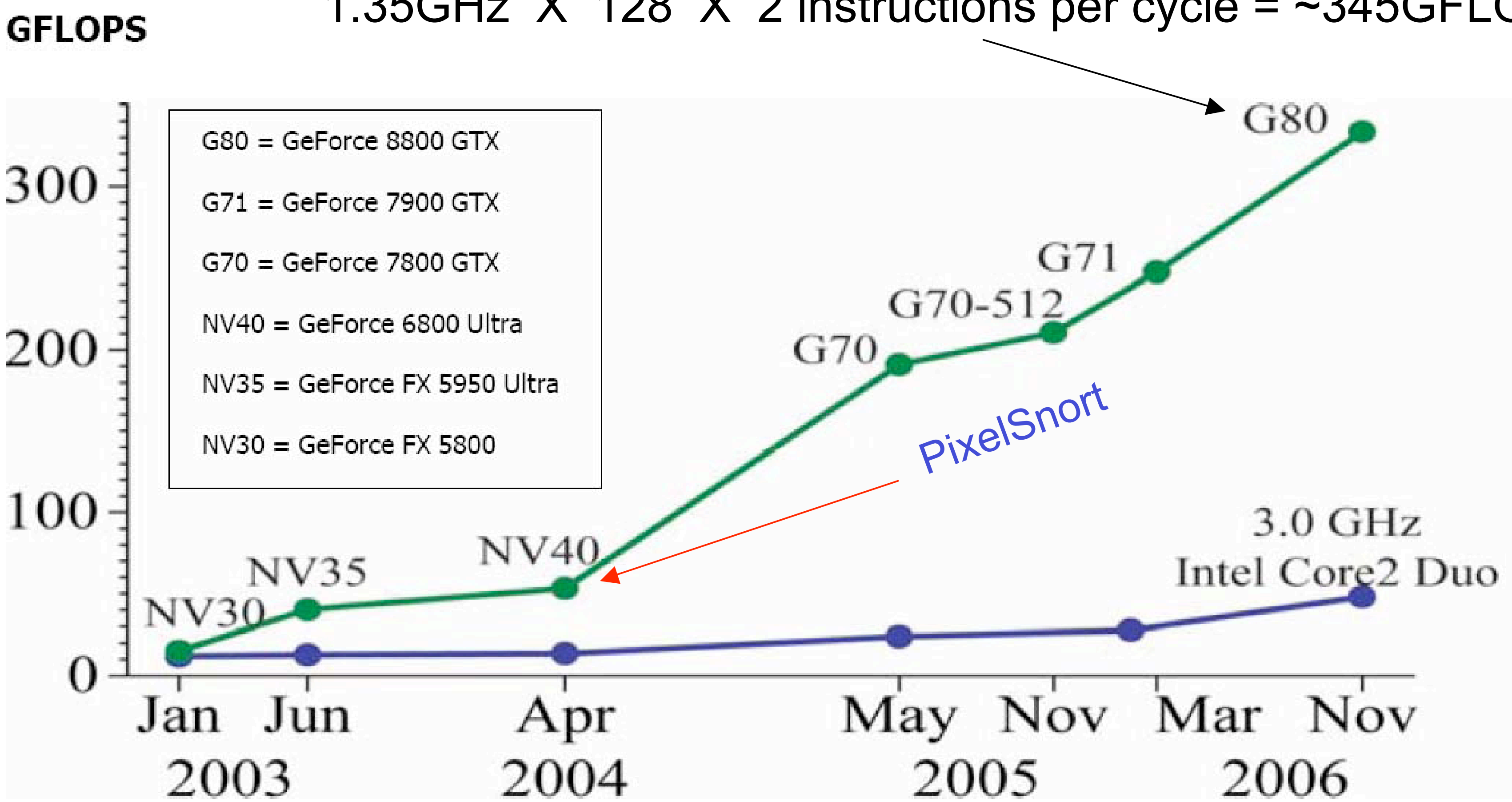


GPUs?

- Multithreading mandatory for applications to take advantage of multicore CPUs
- Tendency to increase the number of processor cores rather than shoot for huge increases in clock rate
- So you're going to have to do multithreading anyway
- New GPUs are massively parallel, use SIMD, thread-based programming model
- Extend threading models to include GPUs as well?
- Yes.   Why?

GPU Horsepower

1.35GHz X 128 X 2 instructions per cycle = ~345GFLOPS



Filling the Gap: GPUs?

- Previous Generation
  - Specialized processors
    - Vertex shaders
    - Fragment shaders
  - Difficult to program
  - Must cast programs in graphical terms
  - Example: PixelSnort (ACSAC 2006)
- Current Generation
  - Uniform architecture
  - Specialized hardware for performing texture operations, etc. but processors are essentially general purpose

THE UNIVERSITY of NEW ORLEANS

NVIDIA G80: Massively Parallel Architecture

~350 GFLOPS per card

Can populate a single box with Multiple G80-based cards

Constraints: multiple PCI-E 16 slots, heat, power supply

8800GTX / G80 GPU

768MB Device Memory

16 "multiprocessors" X 8 stream processors

Total 128 processors, 1.35GHz each

Hardware thread management, can schedule millions of threads

Separate device memory

DMA access to host memory

“Deskside” Supercomputing

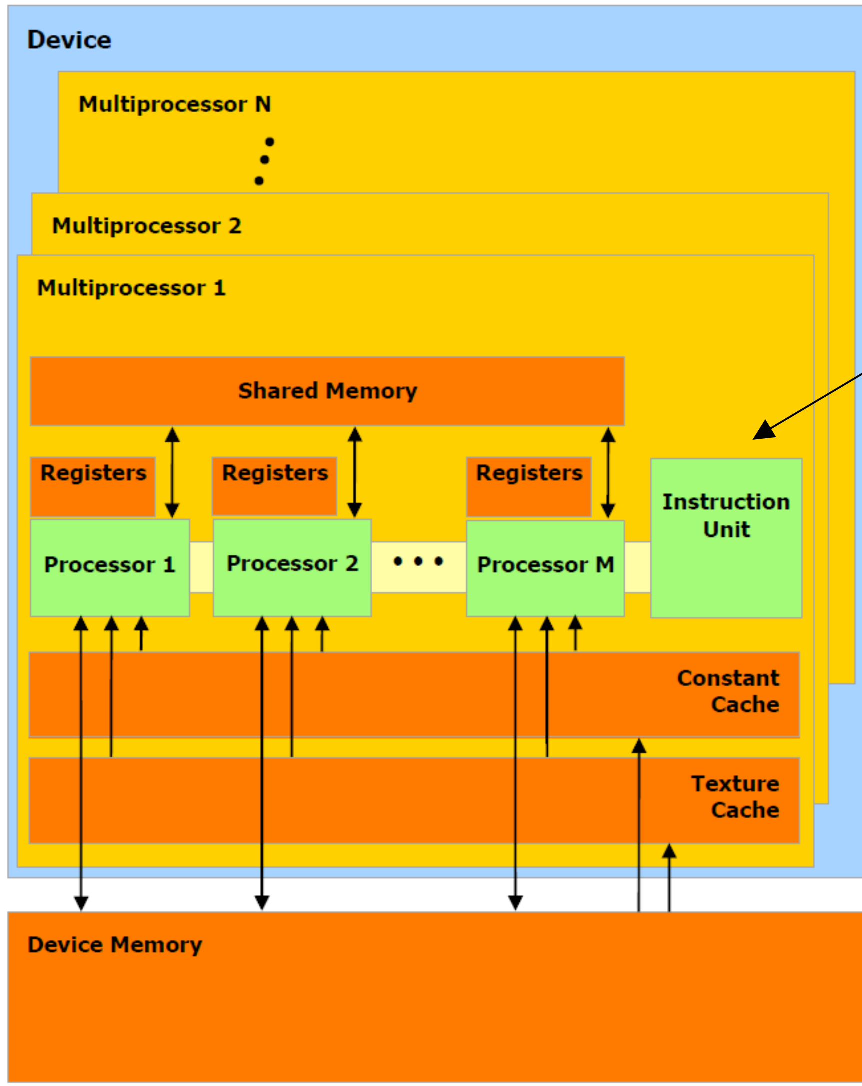
Dual GPUs

3 GB RAM

1 TFLOP

Connects via PCI-E

G80 High-level Architecture

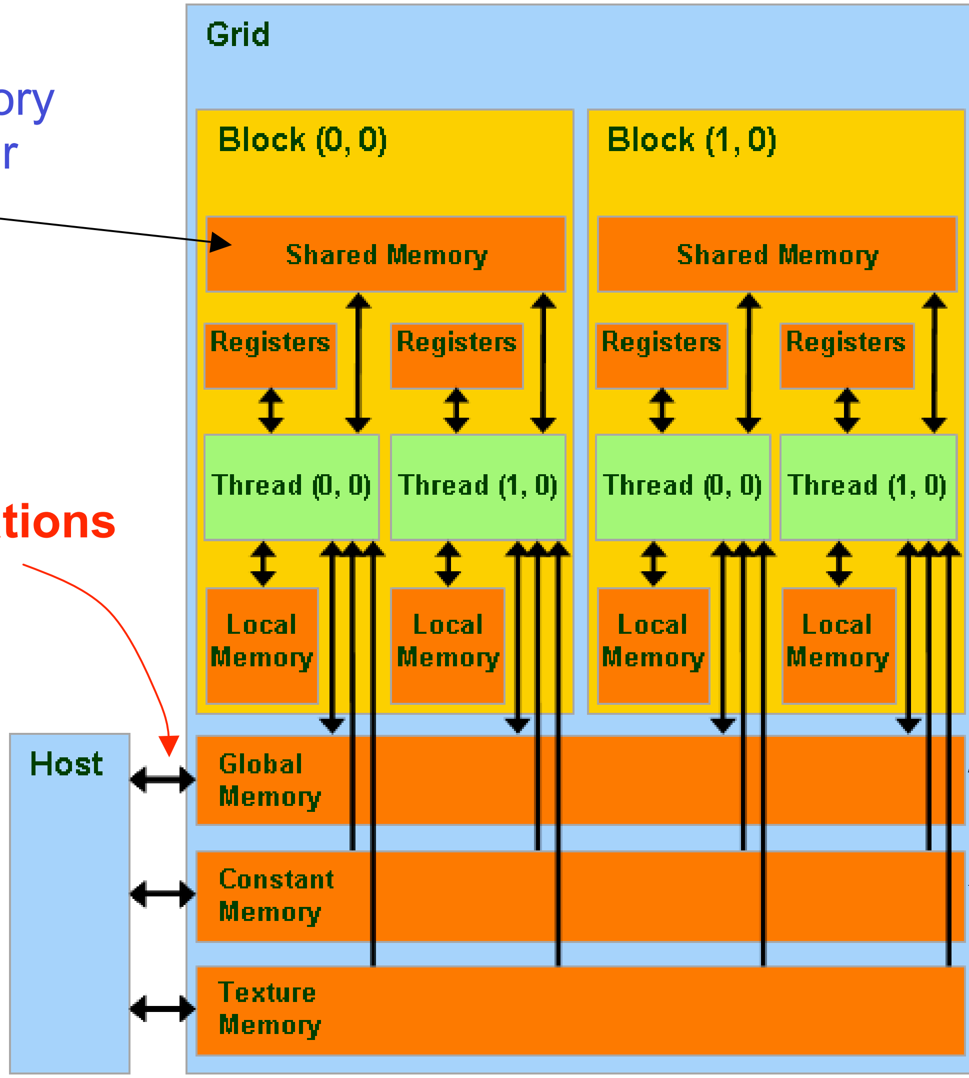


Shared instruction unit is reason that SIMD programs are needed for max speedup

G80 Thread Block Execution

16K shared memory per multiprocessor

Host ←→ Device transfer is main bottleneck for forensics applications



Un-cached device memory (slower but lots of it)

64K of constant memory, 8K cache per multiprocessor

NVIDIA CUDA

- **C**ommon **U**nified **D**evice **A**rchitecture
- See the SDK documentation for details
- Basic idea:
  – Code running on host has few limitations
    - Standard C plus functions for copying data to and from the GPU, starting kernels, …
  – Code running on GPU is more limited
    - Standard C w/o the standard C library
    - Libraries for linear algebra / FFT / etc.
    - No recursion, a few other rules
    - For performance, need to care about thread divergence (SIMD!), staging data in appropriate types of memory

THE UNIVERSITY of NEW ORLEANS

Overview of G80 Experiments

- Develop GPU-enhanced version of Scalpel
- Target binary string search for parallelization
  - Used in virtually all forensics applications
- Compare GPU-enhanced version to:
  - Sequential version
  - Multicore version
- Primary question: Is using the GPU worth the extra programming effort?
- Short answer: Yes.

GPU Carving 0.2

first pass over image file

- Store Scalpel headers/footer DB in constant memory (initialized by host), once
- Loop
  - Read 10MB block of disk image
  - Transfer 10MB block to GPU
  - Spawn 512 * 128 threads
  - Each thread responsible for searching 160 bytes (+ overlap) for headers/footers
    - **Simple binary string search**
  - Matches encoded in 10MB buffer
    - Headers: index of carving rule stored at match point
    - Footers: negative index of carving rule stored at match point
  - Results returned to Host
- End Loop

GPU Carving 0.2: 20GB/Opteron

TABLE I
RESULTS FOR CARVING 20GB DISK IMAGE ON DUAL PROCESSOR, DUAL CORE SUN ULTRA 40 (2.6GHZ AMD OPTERON 2218 PROCESSORS, 16GB RAM). 30 FILE TYPES, ~3M FILES CARVED. EACH RESULT IS THE AVERAGE OF MULTIPLE, SEQUENTIAL RUNS.

Scalpel 1.60 "vanilla"	2672 secs
Scalpel 1.60 "new q"	1784 secs
Scalpel 1.70MT-multicore	1054 secs
Scalpel 1.70MT-gpu-0.20	860 secs

GPU Carving 0.2: 100GB/Opteron

TABLE II

RESULTS FOR CARVING 100GB DISK IMAGE ON DUAL PROCESSOR, DUAL CORE SUN ULTRA 40 (2.6GHZ AMD OPTERON 2218 PROCESSORS, 16GB RAM). 30 FILE TYPES, ~15M FILES CARVED. EACH RESULT IS THE AVERAGE OF MULTIPLE, SEQUENTIAL RUNS.

Scalpel 1.60 "vanilla"	13067 secs
Scalpel 1.60 "new q"	8725 secs
Scalpel 1.70MT-multicore	4958 secs
Scalpel 1.70MT-gpu-0.20	5185 secs

Cage Match!

(Or: The Chair Wants His Machine Back…)

Vs.

Dual 2.6GHz Opteron (4 cores)
16GB RAM, SATA
Single 8800GTX

Single 2.4GHz Core2Duo (2 cores)
4GB RAM, SATA
Single 8800GTX

GPU Carving 0.3

- Store Scalpel headers/footers in constant memory (initialized by host)
- Loop
  - Read 10MB block of disk image
  - Transfer 10MB block to GPU
  - **Spawn 10M threads (!)**
  - Device memory staged in 1K of shared memory per multiprocessor
  - Each thread responsible for searching for headers/footers **in place** (no iteration)
  - **Simple binary string search**
  - Matches encoded in 10MB buffer
    - Headers: index of carving rule stored at match point
    - Footers: negative index of carving rule stored at match point
  - Results returned to Host
- End Loop

first pass over image file

THE UNIVERSITY of NEW ORLEANS

GPU Carving: 20GB/Dell XPS

TABLE III

RESULTS FOR CARVING 20GB DISK IMAGE ON SINGLE PROCESSOR, DUAL CORE DELL XPS 710 (2.4GHz CORE2DUO PROCESSOR, 4GB RAM). 30 FILE TYPES, ~3M FILES CARVED. EACH RESULT IS THE AVERAGE OF MULTIPLE, SEQUENTIAL RUNS.

Scalpel 1.60 "new q"	1260 secs
Scalpel 1.70MT-multicore	861 secs
Scalpel 1.70MT-gpu-0.20	686 secs
Scalpel 1.70MT-gpu-0.30	446 secs

GPU Carving: 100GB/Dell XPS

TABLE IV
RESULTS FOR CARVING 100GB DISK IMAGE ON SINGLE PROCESSOR, DUAL CORE DELL XPS 710 (2.4GHZ CORE2DUO PROCESSOR, 4GB RAM). 30 FILE TYPES, ~15M FILES CARVED. EACH RESULT IS THE AVERAGE OF MULTIPLE, SEQUENTIAL RUNS.

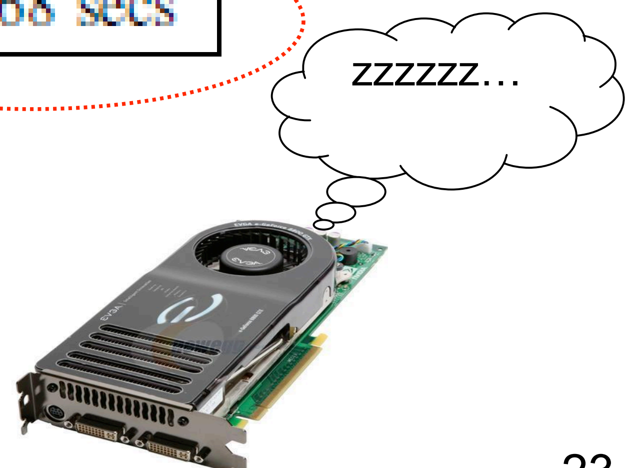
Scalpel 1.60 "new q"	7105 secs
Scalpel 1.70MT-multicore	5096 secs
Scalpel 1.70MT-gpu-0.20	4192 secs
Scalpel 1.70MT-gpu-0.30	3198 secs

THE UNIVERSITY of NEW ORLEANS

Bored GPU == Poor Performance

TABLE V
RESULTS FOR CARVING 500GB DISK IMAGE ON SINGLE PROCESSOR, DUAL CORE DELL XPS 710 (2.4GHZ CORE2DUO PROCESSOR, 4GB RAM). 2 FILE TYPES, ~73,000 FILES CARVED.

Scalpel 1.60 “new q”	9946 secs
Scalpel 1.70MT-multicore	9922 secs
Scalpel 1.70MT-gpu-0.30	12168 secs



But this is NOT an appropriate model for using GPUs, anyway…

Discussion

- Host ←→ GPU transfers have significant bandwidth limitations
  - ~1.3GB/sec transfer rate (observed)
  - 2GB/sec (theoretical)
  - 3GB/sec (theoretical) with page “pinning” (**not** observed by us!)
- Current: Host threads blocked when GPU is executing
  - Host thread(s) should be working…
  - We didn’t overlap host / GPU computation because we wanted to measure GPU performance in isolation
- Current: No overlap of disk I/O and compute
  - For neither GPU nor multicore version
- Current: No compression for host ←→ GPU transfers
- But…

Discussion (2)

- BUT:
  - GPU is currently using simple binary string search
  - Sequential/multicore code using optimized Boyer-Moore string search
- Despite this, GPU much faster than multicore when there's enough searching to do…
- Considering only search time, GPU > 2X faster than multicore even with these limitations

Discussion: 20GB

- Sequential:
  - Header/footer searches: 73%
  - Image file disk reads: 19%
  - Other: 8%
- Multicore:
  - Header/footer searches: 48%
  - Image file disk reads: 44%
  - Other: 8%
- GPU:
  - Total time spent in device <--> host transfers: 7%
  - Total time spent in header/footer searches: 24%
  - Total time spent in image file disk reads: 43%
  - Other: 26%

Conclusions / Future Work

- New GPUs are fast and worthy of our attention
- Not that difficult to program, but requires a different threading model
- Host ←→ GPU bandwidth is an issue
- Overcome this by:
  - Overlapping host and GPU computation
  - Overlapping disk I/O and GPU computation
  - Disk, multicore, GPU(s) should all be busy
  - Overlapping transfers to one GPU while another computes!
  - Compression for host ←→ GPU transfers
- Interesting issues in simultaneous use
  - Simple example: Binary string search: GPU better at **NOT finding things!**
  - **Reduces thread control flow divergence**

Je suis fini, Happy GPU Hacking…

?

Scalpel v1.7x (alpha) is available for testing

Must have NVIDIA G80-based graphics card

Currently runs only under Linux (waiting for CUDA gcc support under Win32)

Feel free to use this as a basis for development of other GPU-enhanced tools…

golden@cs.uno.edu