GeMTC: GPU Enabled Many Task Computing

Scott Krieder, Ben Grimmer, Ioan Raicu
DataSys Laboratory

ILLINOIS INSTITUTE
OF TECHNOLOGY

DataSys
Data-Intensive Distributed
Systems Laboratory

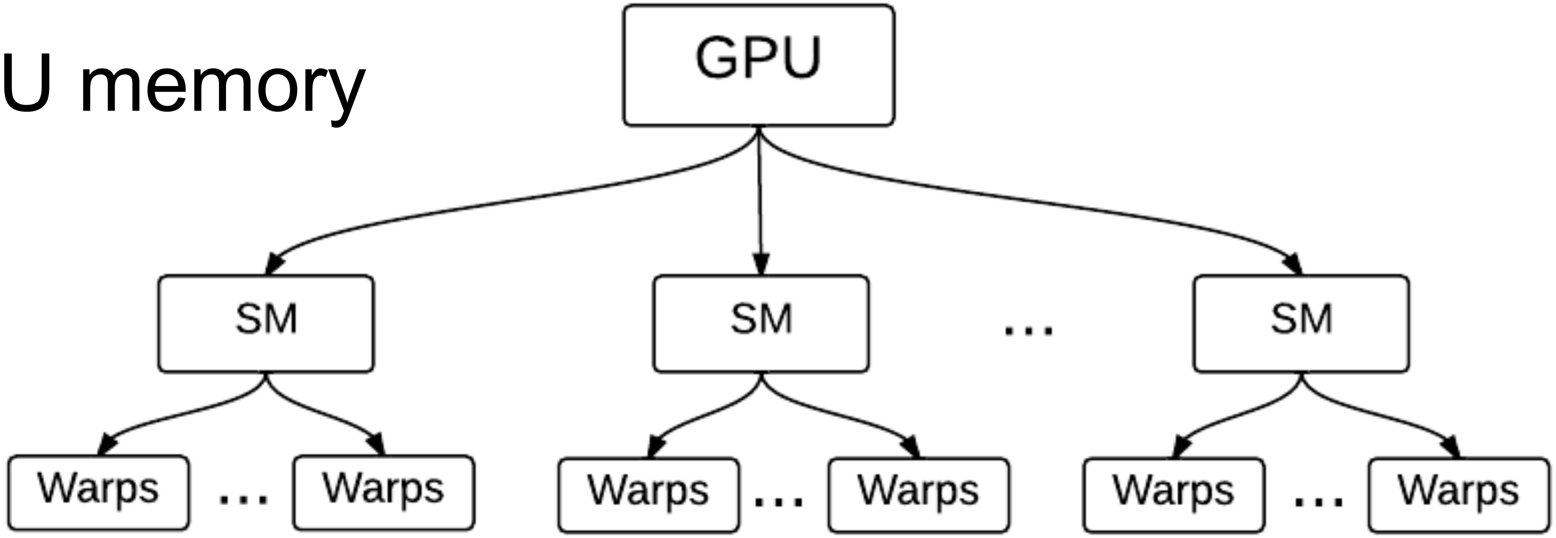
Motivation

Why do we need a framework?

- Default CUDA not efficient for MTC
  - High overheads per CUDA application (100 msec)
  - Multiple applications must timeshare, not concurrent
  - disadvantages with cudaMalloc()

Framework Design/Contributions

- Designed to support MTC workloads
- Manage device on a warp level
- Communicate between CPU and GPU through GPU memory



Framework Design/Contributions

- Much higher granularity
  - 32 thread warps (SIMD worker)
- Improved Dynamic Memory Management
  - CUDA: 110 usec to cudaMalloc() and cudaFree(), not constant cudaMalloc()
  - GEMTC:14 usec to gemtcMalloc() and gemtcFree(), O(1) malloc

Evaluation

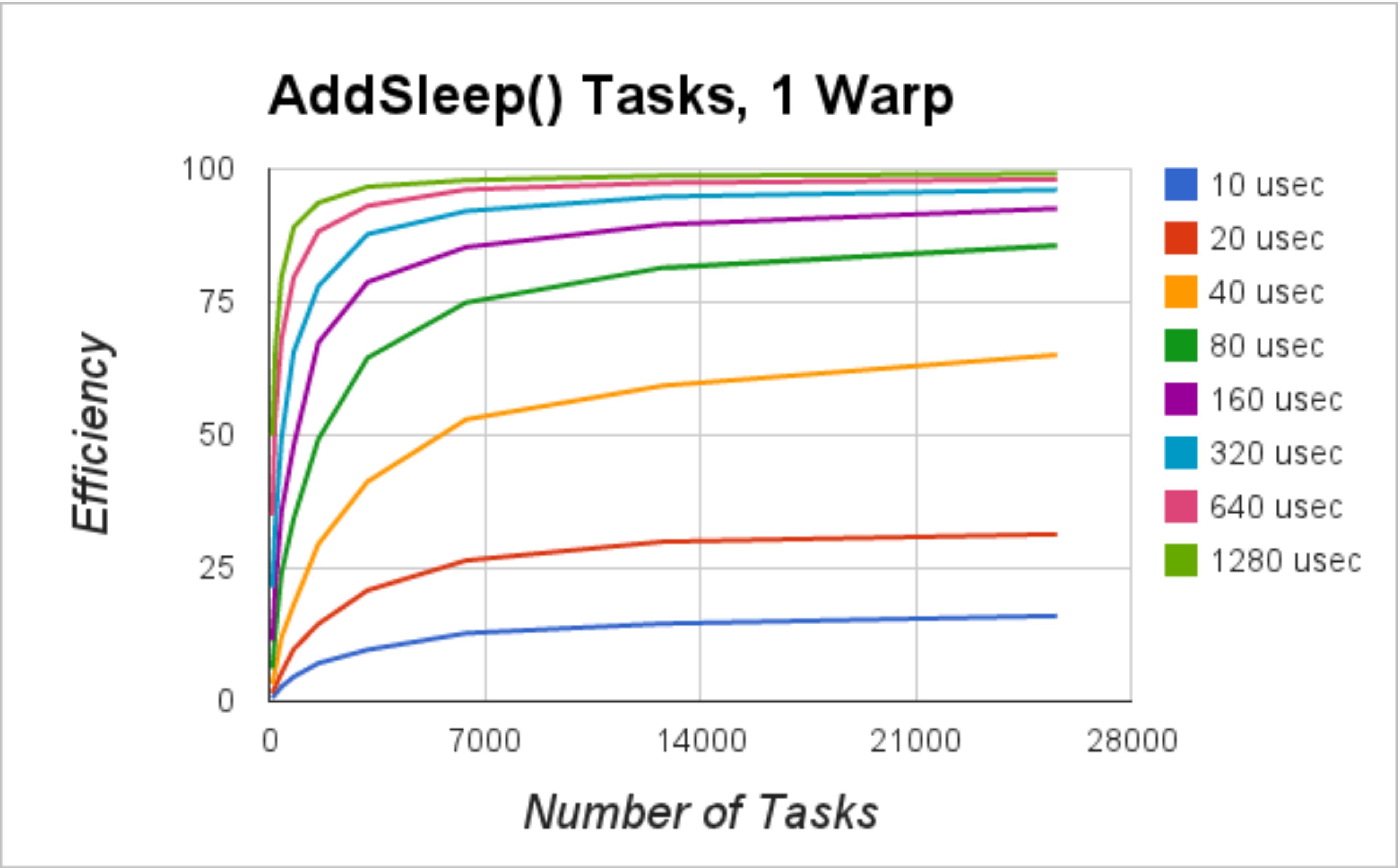
Hardware

- Single Node
  - AMD 6-core CPU
  - 16 GB of RAM
- NVIDIA GTX-670
  - 2GB GDDR5
  - 84 warps

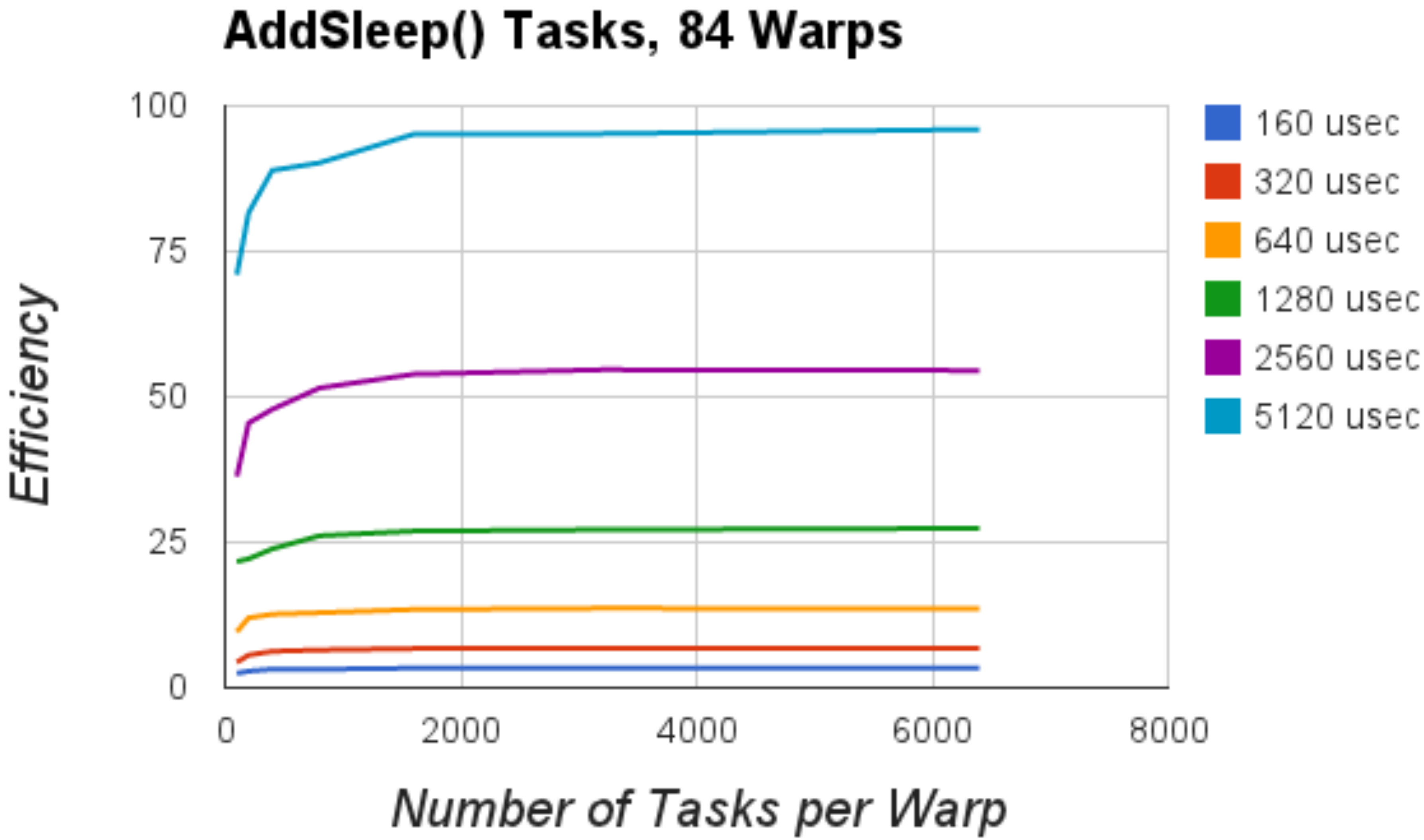
Micro-Kernels

- Sleep()
  - busy wait
- Matrix-Square()

Results


AddSleep() Tasks, 1 Warp
Efficiency
100
75
50
25
0
0
7000
14000
21000
28000
Number of Tasks
10 usec
20 usec
40 usec
80 usec
160 usec
320 usec
640 usec
1280 usec

Results


AddSleep() Tasks, 84 Warps
Efficiency
100
75
50
25
0
0
2000
4000
6000
8000
Number of Tasks per Warp
160 usec
320 usec
640 usec
1280 usec
2560 usec
5120 usec

Future Work

- Integrate with Swift/T
- Coalescing memory copy for Tasks
- Run MTC workloads on Intel MIC
- Evaluate GEMTC on GPU Simulators
- Port GEMTC to OpenCL

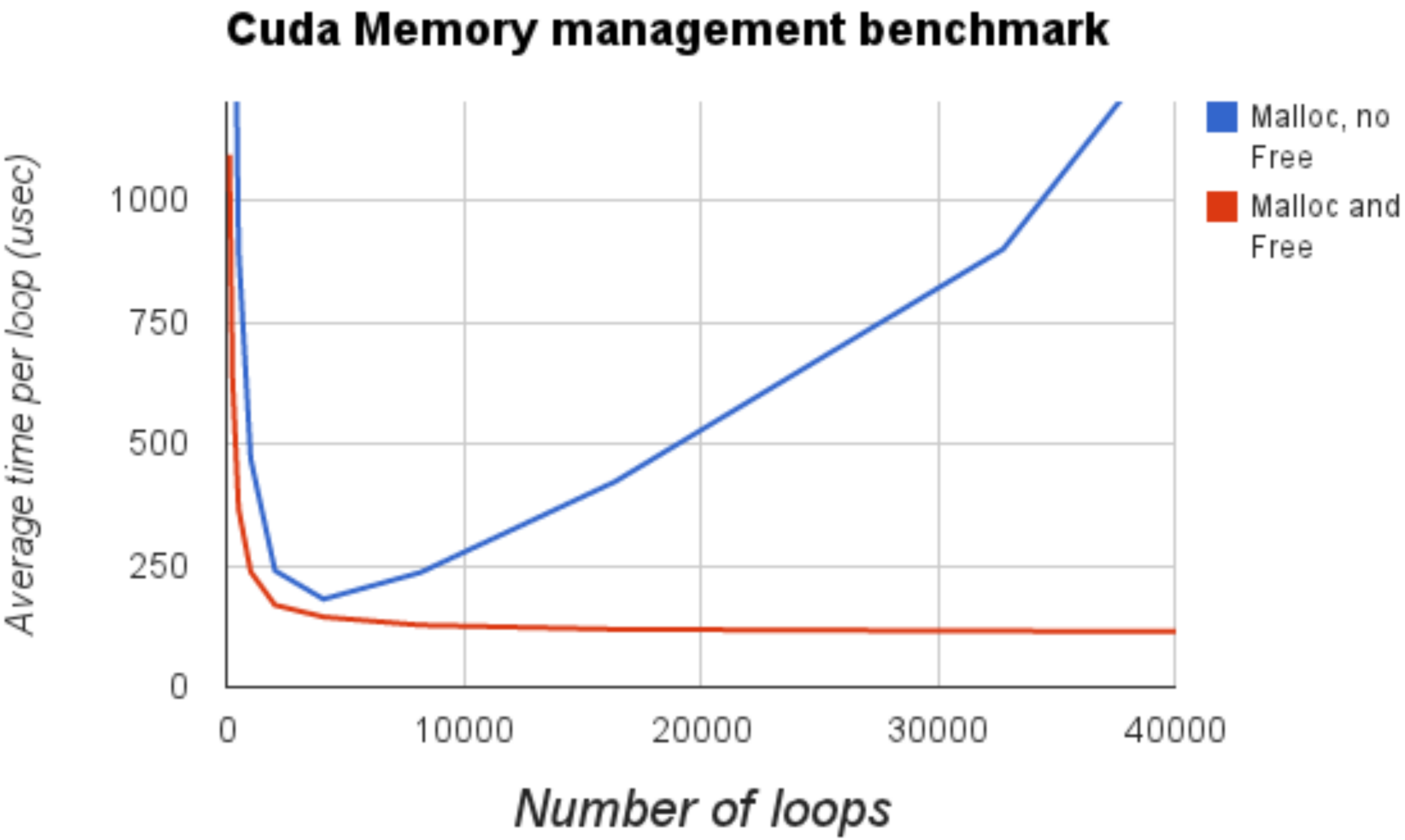
Questions

Scott Krieder
skrieder@iit.edu

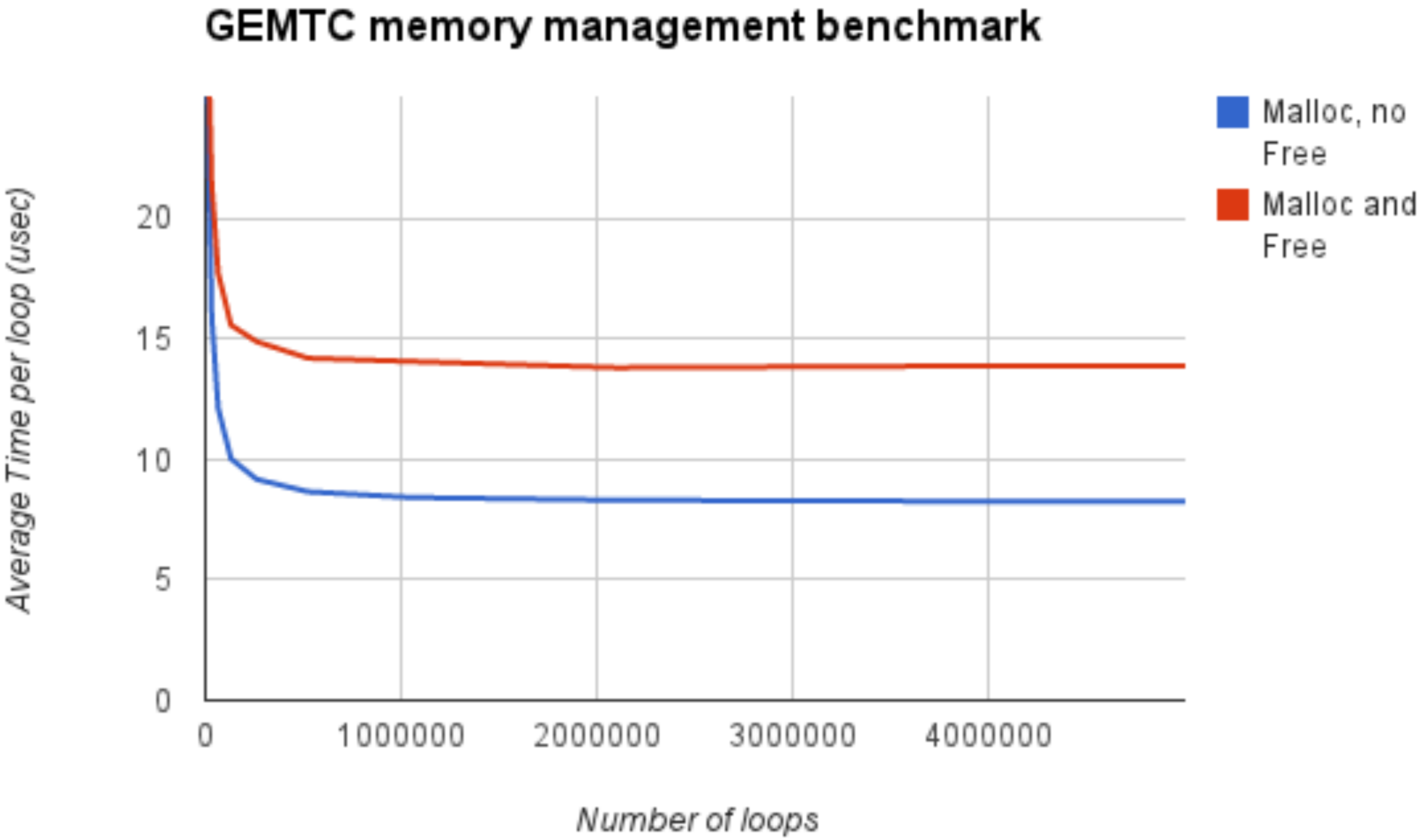
Benjamin Grimmer
bgrimmer@hawk.iit.edu

Ioan Raicu
iraicu@cs.iit.edu

cudaMalloc() Performance



gemtcMalloc() Performance


GEMTC memory management benchmark
Average Time per loop (usec)
Number of loops
🟦 Malloc, no Free
🟥 Malloc and Free