Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU

Omar Navarro Leija and Richard Zang

Throughput Computing Applications

What are throughput computing applications?

- Applications that handle “big data”.
- Applications that feature data level parallelism.
- Applications that can process data independently and out of order.
- Applications that should complete in a timely manner.

Why do we need such applications?

- The authors predicted in 2009 that there would soon be exabytes (10^18 bytes) of data in the world.
- According to Northeastern, in 2016, we produce 2.5 exabytes every single day.

Overview

Main Points of Paper

“We reexamine a number of claims, that GPUs perform 10X to 1000X better than CPUs on a number of throughput kernels/applications.”

“After tuning the code for BOTH CPU and GPU, we find the GPU only performs 2.5X better than CPU.”

“This puts CPU and GPU roughly in the same performance ballpark for throughput computing.”

Performance Results

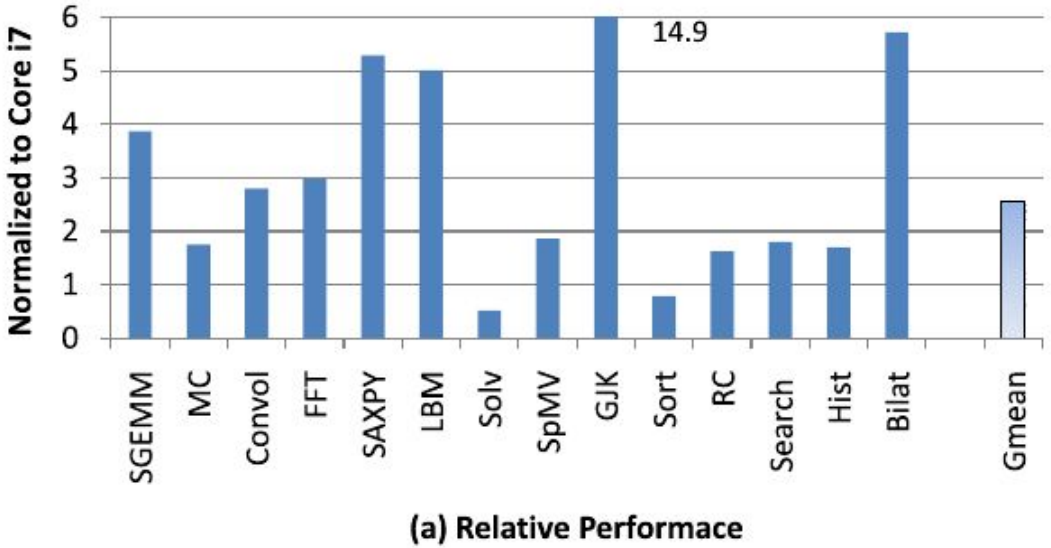

Figure 1: Comparison between Core i7 and GTX280 Performance.

Hardware Comparison

	Num. PE	Frequency (GHz)	Num. Transistors	BW (GB/sec)	SP SIMD width	DP SIMD width	Peak SP Scalar FLOPS (GFLOPS)	Peak SP SIMD Flops (GFLOPS)	Peak DP SIMD Flops (GFLOPS)
Core i7-960	4	3.2	0.7B	32	4	2	25.6	102.4	51.2
GTX280	30	1.3	1.4B	141	8	1	116.6	311.1/933.1	77.8

Benchmarks

Benchmarks

Kernel	Application	SIMD	TLP	Characteristics
SGEMM **(SGEMM) [48]**	Linear algebra	Regular	Across 2D Tiles	Compute bound after tiling
Monte Carlo **(MC) [34, 9]**	Computational Finance	Regular	Across paths	Compute bound
Convolution **(Conv) [16, 19]**	Image Analysis	Regular	Across pixels	Compute bound; BW bound for small filters
FFT **(FFT) [17, 21]**	Signal Processing	Regular	Across smaller FFTs	Compute/BW bound depending on size
SAXPY **(SAXPY) [46]**	Dot Product	Regular	Across vector	BW bound for large vectors
LBM **(LBM) [32, 45]**	Time Migration	Regular	Across cells	BW bound
Constraint Solver **(Solv) [14]**	Rigid body physics	Gather/Scatter	Across constraints	Synchronization bound
SpMV **(SpMV) [50, 8, 47]**	Sparse Solver	Gather	Across non-zero	BW bound for typical large matrices
GJK **(GJK) [38]**	Collision Detection	Gather/Scatter	Across objects	Compute Bound
Sort **(Sort) [15, 39, 40]**	Database	Gather/Scatter	Across elements	Compute bound
Ray Casting **(RC) [43]**	Volume Rendering	Gather	Across rays	4-8MB first level working set, over 500MB last level working set
Search **(Search) [27]**	Database	Gather/Scatter	Across queries	Compute bound for small tree, BW bound at bottom of tree for large tree
Histogram **(Hist) [53]**	Image Analysis	Requires conflict detection	Across pixels	Reduction/synchronization bound
Bilateral **(Bilat) [52]**	Image Analysis	Regular	Across pixels	Compute Bound

Platform Optimization Guide

CPU Optimizations

Kernels can linearly scale with number of cores. Use cache blocking to fit working set in cache.

Cache blocking: “The key idea behind blocking is to exploit the inherent data reuse available in the application by ensuring that data remains in cache across multiple uses.” -Intel https://software.intel.com/en-us/articles/cache-blocking-techniques

Fortran Source Example:

```
do j=1,N
  do k = 1,N
    do i = 1,N
      c(i,j) = c(i,j) + a(i,k) * b(k,j)
    end do
  end do
end do
```

Modified Fortran Source:

```
do JJ = 1, N, TJ

  do KK = 1, N, TK
    do jjj = 1,min(tj,N-jj+1)                    ! BCOPY - no transpose
      do kkk = 1, min(tk,N-kk+1)
        p(kkk,jjj-1+1) = B(kk+kkk-1, jj+jjj-1)
      end do
    end do
    do II = 1, N, TI
      do iii = 1,
        min(ti,N-ii+1)                    !ACOPY - transpose
        do kkk = 1, min(tk,N-kk+1)
          Q(kkk,iii) = A(ii+iii-1, kk+kkk-1)
        end do
      end do
      do J = 1, min(tj,N-jj+1), 4
        do I = 1, min(ti,N-ii+1), 2
          t1 = 0 ; t2 = 0 ; t5 = 0 ; t6 = 0 ; t9 = 0 ; t10 = 0 ; t13 =0 ;
          !DIR$ vector aligned                    !DIR$ unroll(2)
          do K = 1,min(TK,N-kk+1)      ! Innermost loop, vectorized and un
            qi = Q(K,I)           ;    qi1 = Q(K,I+1)
            t1 = t1+qi*P(K,J)     ;    t2 = t2+ qi1*P(K,J)
            t5 = t5+ qi*P(K,J+1)  ;    t6 = t6+ qi1*P(K,J+1)
            t9 = t9+ qi*P(K,J+2)  ;    t10 = t10+ qi1*P(K,J+2)
            t13 = t13+ qi*P(K,J+3);    t14 = t14+qi1*P(K,J+3)
          end do
          c(i+ii-1,j+jj-1) = c(i+ii-1,j+jj-1) +t1          ; c(i+1+ii-1,j+j
          c(i+ii-1,j+1+jj-1) = c(i+ii-1,j+1+jj-1) + t5     ; c(i+1+ii-1,j+1
          c(i+ii-1,j+2+jj-1) = c(i+ii-1,j+2+jj-1) + t9     ; c(i+1+ii-1,j+2
          c(i+ii-1,j+3+jj-1) = c(i+ii-1,j+3+jj-1) + t13    ; c(i+1+ii-1,j+3
        end do
      end do
    end do
  end do
end do
```

GPU Optimization

Global inter-thread synchronization is very costly! Requires kernel termination and launch overhead. For some benchmarks, they used “constraint reordering to minimize conflicts among neighboring constraints, which are global synchronization points.”

Optimization: Use the shared buffers to reduce bandwidth consumption.

Hardware Recommendations

High Compute Flops And Memory Bandwidth

To increase flops:

- Increase core count. This increase is limited by the die area.
- Increase SIMD width. This increase is not limited by die area but needs regular memory access.

To increase memory bandwidth:

- Increase memory capacity; this is limited by the costs of memory compared to the bulk capacity we see with CPU RAM.
- Develop better memory technology.

Large Cache

Each streaming multiprocessor has its own cache which should be able to hold the entire working set of data.

Working set size can scale with the data set, the number of cores or threads, and cache sizes will need to grow to accommodate future workloads.

Gather/Scatter

The authors hypothesize that gather/scatter support can greatly increase the speed of certain workloads.

To enable gather/scatter:

- Implement in hardware, although this would be difficult
- Implement in software using memory banks
- Use shuffle instructions to share data

Efficient Synchronization and Cache Coherence

Better synchronization is needed for CPUs and GPUS because current solutions do not scale well with additional cores and increased SIMD width.

Barrier costs can become less trivial as the number of cores increase and the workload per thread decreases.

Fixed Functional Units

Create fixed functional units to accelerate commonly used operations.

- Transcendental operations
- Texture mapping/sampling units
- Encryption/Decryption

History & Controversy

Changes in Hardware Since 2009

Intel i7 960
Clock Speed: 3.2 GHz
Cores: 4
L2 Cache: 4x 256 KB
L3 Cache: 8MB
Released: October 2009

GTX 280
SM Cores: 8
SIMD Lane Width: 8
CUDA Cores: 280
L2 Cache: 1 MB
Memory Bandwidth: 141.7 GB/sec
1 GB Memory
Released: June 2008

Intel i7 7700
Clock Speed: 3.6 GHz
Cores: 4
L2 Cache: 4x 256 KB
L3 Cache: 8MB
Released: January 2017

GTX 1080
SM Cores: 20
SIMD Lane Width: 16
CUDA Cores: 2560
L2 Cache: 2 MB
Memory Bandwidth: 320 GB/sec
8 GB Memory
Released: May 2016

Nvidia's Response

GPUS ARE ONLY UP TO 14 TIMES FASTER THAN CPUS" SAYS INTEL

"It's a rare day in the world of technology when a company you compete with stands up at an important conference and declares that your technology is *only* up to 14 times faster than theirs."

https://blogs.nvidia.com/blog/2010/06/23/gpus-are-only-up-to-14-times-faster-than-cpus-says-intel/

Criticism

CPU World Article: “Intel: 2-year-old Nvidia GPU Outperforms 3.2GHz Core I7”

The GTX280 had been out for two years when the paper compared them. It has been noted this is Nvidia’s fault for delaying the release of their GTX380.

http://www.pcworld.com/article/199758/Intel_2_year_old_Nvidia_GPU.html
https://arstechnica.com/business/2010/06/intel-scores-own-goal-against-core-i7-in-nvidia-spat/

Questions

Question Time

Could you explain the features compared in Table 2, especially SP SIMD Width and DP SIMD width?

Question Time

What exactly is the difference between SIMD and thread-level parallelism?

Question Time

What is rationale for choosing Intel i7-960 and GTX280? Why is it a fair comparison?

Question Time

Why is the external memory bandwidth of GTX280 greater than i7? Is a GPU's memory bandwidth generally greater than CPU's?

Question Time

When the speedup of GPU is not very significant, is it feasible to divide the work and have the GPU and CPU to work together? Is it scalable?

Question Time

The hardware optimization suggestions seem to heavily rely on the features or characteristics of specific throughput computing kernels. Is this sound design for CPUs and GPGPUs, to have hardware optimization that is application specific?

Question Time

The paper shows that with careful optimization, GPU demonstrates only averaged 2.5x speed up on throughput computing. However, it is widely believed that GPU can be highly suitable for high performance computing. Is it the fact that most of the CPU computing programs are not well optimized? or other metrics of computing other than throughput are crucial?

Question Time

On a general note, CPUs can be overclocked upto 4.1GHz. In such a state, what can be the potential performance difference between CPUs and GPUs (power consumption aside)?

Question Time

In 5.1 the paper mentioned that memory bandwidth on CPUs is low as compared to GPUs. Would better hardware optimization on memory access significantly improve CPUs throughput performance? If so, would CPUs be as capable as GPUs on certain tasks which require intense memory access?

Question Time

In 5.1 the paper mentioned that memory bandwidth on CPUs is low as compared to GPUs. Would better hardware optimization on memory access significantly improve CPUs throughput performance? If so, would CPUs be as capable as GPUs on certain tasks which require intense memory access?

Question Time

Do you think there are high throughput workloads that would allow a CPU to outperform a GPU when synchronization is introduced?

Question Time

Will the speedup rate vary when the test programs are applied to larger scale of data? Should the Gustafson's law be considered?

Question Time

Is using cache bank per thread in a warp a gather/scatter method provided by hardware in GPUs?

Question Time

Could you please elaborate on and illustrate the section that states that Reduction operations do not scale with increasing thread count and data level parallelism?

Question Time

What's the difference between "general shuffle and swizzle instructions" and "gather/scatter operations"? How are they related? It seems like we saw something related to gather/scatter with the transpose example in class (i.e. going through shared memory to write back to global memory densely)?