J.P.Morgan

May 2017

Practical aspects of porting Monte Carlo exotic derivative pricing engines to IBM Power 8+ with P100 GPUs

Richard Hayden, Andrey Zhezherun, Oleg Rasskazov (JP Morgan)

GPUs in JP Morgan

- JP Morgan is extensively using GPUs to speed up risk calculations and reduce computational costs since 2011.
  - Speedup as of 2011 ~ 40x
- Large Cross Asset Quant Library (C++, Cuda)
  - Monte Carlo and PDEs
  - GPU code
    - Hand-written Cuda Kernels
    - Thrust
    - Auto-Generated Cuda Kernels
- Hardest part in delivering GPUs to production
  - Bugs

GPU Compute use cases

- Revolutionary Compute density within the node
  - Machine Learning applications
    - E.g. fraud detection
- Fastest End-to-End calculations
  - Real-time risk
- **Reducing Cost of Compute**
  - Focus of the talk

Reducing Cost of Compute with GPUs

- Cost of Compute ~ Cost of throughput, Occupancy
  - High overheads on protecting competition-sensitive data
    - Extra cost outlays per node
  - => Like top range chips
  - => Like Compute density, e.g. multiple GPUs per node

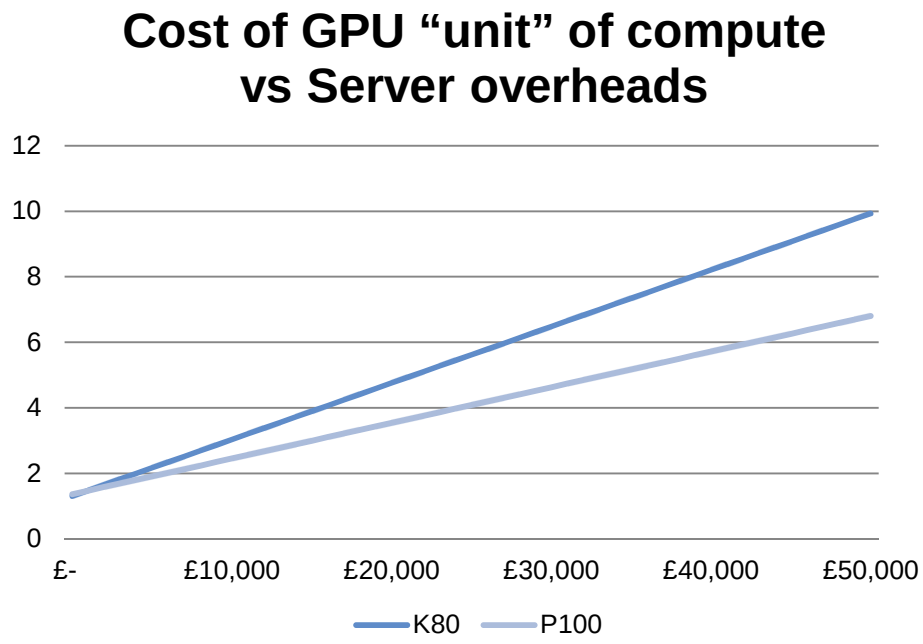
- Dominant Problem in Finance: Monte Carlo for risk calculations
  - Mostly double-precision calculations
  - Embarrassingly data parallel
  - Memory hungry
    - Caching opportunities improve throughput
    - Caching is vital to match CPU throughput
  - “Heterogeneous” compute pricing tasks, e.g. pricing different instruments with different models*
  - Compute growths at “nb of tasks” level, not task compute complexity level*

Sample TCO model

- 2 GPU server
  - K80 £3,800 (Sample internet price) 2.9TF
  - P100 PCIe £6,400 (Sample internet price) 4.6TF
- Cost per Performance, normalized:

$$\frac{2 * GPU\ Price + Server\ Overheads}{2 * GPU\ Peformance}$$

- Server overheads =
  - Power consumptions
  - Blade Hardware cost
  - Data center costs
  - Software licensing
  - Etc
- **Summary:**
  - Top-range chips are better
  - Higher GPU density is better

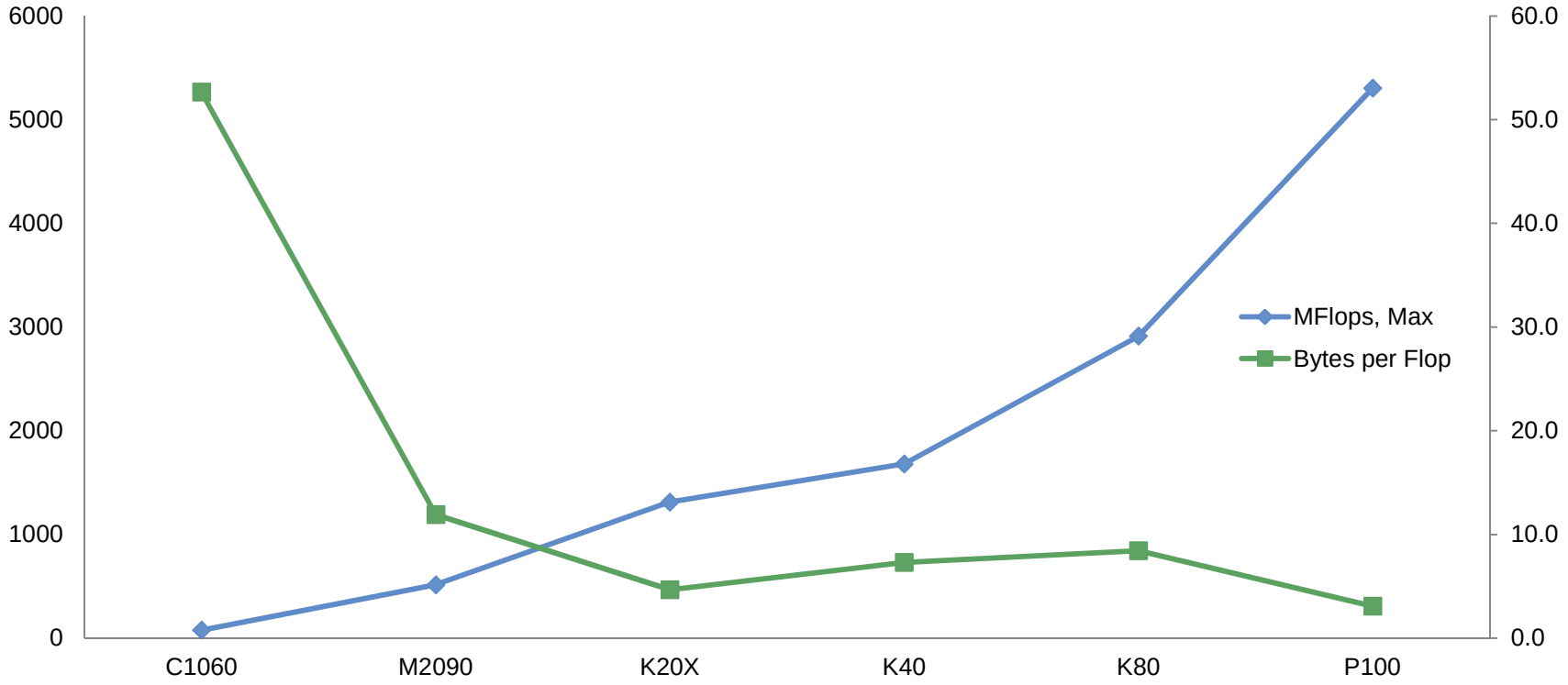


GPU as a compute throughput device

- Summary:
  - Double precision
  - Embarrassingly data parallel with “small” tasks
    - Over the years GPUs speed up faster than typical task size
  - High occupancy
    - Mixes serial / parallelisable codes
      - Example Tweak and reprice for multiple Greeks
    - Multi-tenancy on a single GPU
  - GPU memory size limitations
    - Caching / Calc reuse is vital in lowering cost of compute
  - Compute density
    - Many GPU cards per blade

GPU as a compute throughput device

- Multi-tenancy on a single GPUs is ever more important
  - Task stays “roughly” the same, but number of tasks grow
  - Compute roughly doubles every 2 years
- Number of GPUs per box increases, driving TCO down
- Ratio of GPU Memory/GPU Compute massively decreasing over time



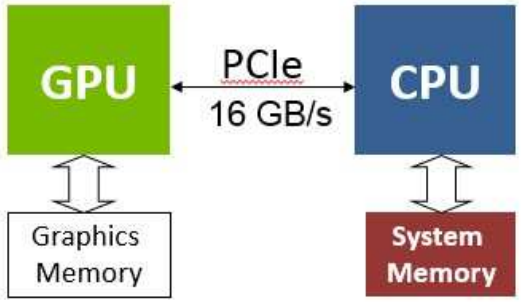
J.P.Morgan

IBM Power 8+ with P100 GPUs

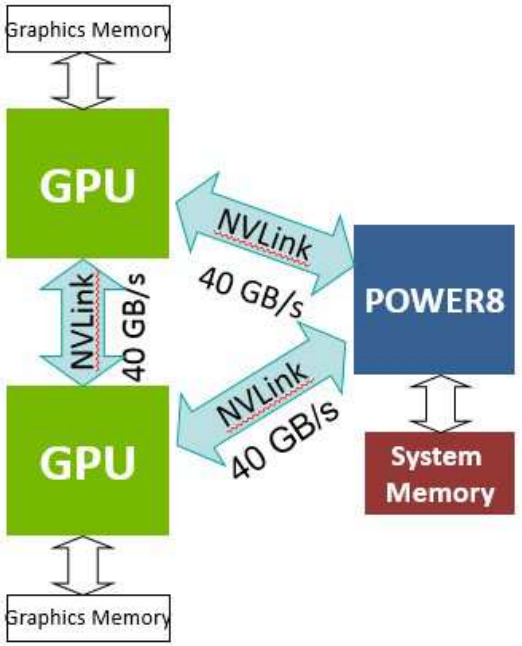
- Half of the server (one chip).

  From https://www.ibm.com/blogs/systems/ibm-power8-cpu-and-nvidia-pascal-gpu-speed-ahead-with-nvlink/

2.5x Faster CPU-GPU Connection via NVLink



CPU-GPU Systems Connected via PCI-e



NVLink Enables Fast Unified Memory Access between CPU & GPU Memories

“Ideal” GPU throughput compute box?

- IBM Power System S822LC !
  - Fast CPU <-> GPU interconnect
    - 2 NVLink bricks for connection
    - 40 GB/s one direction, 80 GB/s bi-directional
  - 4 x P100 in a server
    - Twice the GPU density of what we use in production
  - 2 x Power 8+ chips @ 4GHz

- More than 4 GPUs = lower CPU-GPU interconnect speed ?

POC with IBM

- Evaluate IBM Power System S822LC as a candidate for production GPU compute based on
  - Monte Carlo benchmarks for P100 systems
    - STAC-A2
  - High GPU density in the server
  - NVLink interconnect between CPU and GPUs
    - Only platform to support it
    - In future: NVLink 2.0
    - Opens up a possibility to efficiently use GPUs for a wider class of algorithms
- Interesting early results, but work in progress

Testing GPU throughput

- Quant library rebuilt for IBM Power
- Production-like test harness simulating compute load on the box
  - Strips out “network communication / task load effects”
  - Fully loads the box
  - Represents a mixture GPU production trades for various asset classes
  - GPU scaling 1– 4
  - Slot per GPU scaling 1 – N threads utilizing GPU (up to 10 in experiments)
- CUDA 8
- NVProf + Library measurements
- No P100 specific optimizations
- Comparison only against Intel Haswell + K80s

Testing IBM Power System S822LC

- Case 1: Compute bound use cases
  - Monte Carlo GPU kernels are 3.6x faster end to end (vs ½ of K80)
    - Before P100 specific optimizations
  - 2x GPU density vs K80 boxes currently in production
  - Case further enhanced by TCO model considerations

Testing IBM Power System S822LC

- Cases 2 and 3: “Chatty CPU/GPU code”
  - Latency bound calculations
    - “cpu-like” flow of memory allocs
    - Aggregate bounds on # calls / s
  - Sublinear scaling on #slots/GPU
    - Not Compute Bound
    - Too many API calls
    - => Code refactoring
  - Sublinear scaling on #GPUs being used
    - 2 x performance drop from using 4 GPUs in the system
    - Anti-scaling for cudaMalloc / cudaFree / cudaMemcpy
      - => Significant code refactoring
      - => Better driver support for “compute” use cases?
      - Switching to custom allocator brings in only ~15% throughput improvement

Trade 1	Throughput per GPU relative to the 1 GPU/1slot				
#GPUs\#slots	1	2	3	4	5
1	1.000	1.742	2.422	2.826	3.099
2	0.968	1.604	2.243	2.793	2.942
4	0.972	1.642	2.292	2.715	2.928

Trade 2	Throughput per GPU relative to the 1 GPU/1slot				
#GPUs\#slots	1	2	3	4	5
1					3.996
2					3.144
4	1.000	1.574	1.921	2.031	2.136

Further work

- Throughput corner cases
  - Reducing number of API calls
    - Custom memory allocators to reduce cudaFree / cudaMalloc
    - Experiment with Unified memory to reduce nb of mem copies
    - Explicitly manage copying data for kernels
- “Oversubscribe” in terms of GPU memory resources
  - Unified memory to manage the transfers
  - Further Leverage NVLink to CPU
- Reducing caching
  - Replacing caching with compute
  - Caching in single precision?
- Re-architecting Quant library to run Greek calculations in parallel on GPUs
  - Hard

Acknowledgements

- Francois Thomas, IBM
- Jiri Kraus, NVidia

Summary

- Our throughput compute use cases bottleneck on GPU memory size and API calls latencies
  - Code re-engineering is needed
  - Our GPU programming is harder than it was for C1060
- IBM Power 8+, 4 P100, NVLink – Enterprise system with fast CPU-GPU interconnect (NVLink) that helps to improve GPU memory bound use cases
  - Potentially the sweetest spot for our throughput GPU compute
  - Quant libraries do run on IBM Power 8+
  - On-going work

Case for throughput compute on GPUs

- Fractional GPU use by embarrassingly parallel tasks
  - No inter-gpu communications
  - Reduction of memory per compute on GPU
    - Forcing more API calls / data transfers
    - Only going to get worse for new generations
  - We expect GPU density in the servers to keep rising
- Technology challenges
  - Despite maturity of Unified memory, we need to write own allocators and memcopies to extract performance
  - Sublinear driver API call scaling when multiple GPUs are used