Super GPU & Super Kernels: Make programming of multi-GPU systems easy

Michael Frumkin, May 8, 2017
AGENDA

Why super GPU is needed
Extending CUDA view into clusters
Example: Sparse Matrix Vector Multiplication
Implications for SW and HW
Example: FFT
Example: Caffe AlexNet
Take-outs
MULTI-GPU SYSTEMS

Multi-GPU nodes are here and more coming

- DGX1 - 8 GPUs
- Coming: Summit and Aurora: about 15 K GPUs

GPU enabled clusters

Need orchestrate GPUs computations

GPUs are connected by a network based on NVlinks

Supported by MPI-like library NCCL
WORK RASTERIZATION

// Kernel invocation with 3 x 2 grid of 4 x 3 threads

dim3 grid(3, 2, 1);
dim3 threads(4, 3, 1);
MatrixAdd<<<grid, threads>>>(A, B, C)

Allows to distribute computations

CUDA success in programming massive number of threads can be extended to multi-GPU systems

Tiling GPUs into a super-GPU seems like logical step in scaling
WORK SUPER RASTERIZATION

dim3 sblock(8, 1, 1);
__host__ void SpMVKernelS(
    dim3 sblock, float** d_matr, int** d_idx, ...) {
    #pragma omp parallel for num_threads(sblock.x)
    for (int i = 0; i < sblock.x; ++i) {
        cudaSetDevice(i);
        SpMVKernel<<<grid, threads>>>(d_matr[i], d_idx[i], ...);
    }
}

dim3 sblock(8)

dim3 sblock(4, 3)

dim3 sblock(2, 2, 2)
SPARSE MATRIX-VECTOR MULTIPLICATION (SPMV)

No cross-GPU communications
Super-linear speedup
  8.4 on DGX1
  4.3 on PLX connected K40m
IMPLICATIONS FOR SW AND HW

Driver has an option to recognize super-kernels and optimize launch

- One option: CudaLaunchKernelCooperative
- Move some load needed for kernel launch to GPUs

Vectorize kernel launches

- PCIe supports broadcast

Allocation of page tables can be directed by superblock
FFT SUPERKERNEL

\[ F_{rs} = (F_r \otimes I_s)D_r(I_r \otimes F_s) \]

dim3 sblock(4, 2, 1);
CopyDataToGPUs(sblock, r * s, h_src, d_dst);
ButterflyRightWing(sblock, r, s, d_dst, d_res);
GlobalTranspose<T2>(sblock, r, s, d_res, d_tsr, d_tmp, handle);
ButterflyLeftWing(sblock, r, s, d_tmp, d_dst);
CopyDataFromGPU(sblock, r * s, d_dst, h_res);
TRAINING NEURAL NETWORKS (CAFFE)

Many layers

Big data volumes have to pass through

Most computationally expensive are convolutional layers

Main ops: Gemm, Winograd, FFT

\[ Y_{nkpq} = \sum_{c=1}^{C} \sum_{r=1}^{R} \sum_{s=1}^{S} X_{n, c, p + r, q + s} W_{kcrs} \]

Data parallel distribution requires AllReduce to update weights
In train(), create dim3 sblock = get_gpus();
Refer to sblock.Volume() instead of gpus.size();
Pass sblock to P2Psync constructor:

\[ \text{P2Psync::P2Psync(solver, root_id, sblock, solver->params)}; \]
Use sblock in P2Psync::Run();

Alternative: caffe_gpu_gemm<float>(sblock, gemm_params, ...)

AlexNet Scalability

Number of GPUs

Iterations per second
TAKE-AWAYS

Multi-GPU programming using superblocks is easy

Rasterization of the superkernel is as intuitive as rasterization of CTAs
  • Results in good scalability assuming good load balance and small communications (SpMV)

Transparently distributes work, allows to concentrate on optimization of the communications
  • Pipeline communications and computations

Data distributions can be described by the superbblock
QUESTIONS?

Michael Frumkin
mfrumkin@nvidia.com