MPI, CUDA, and Quantum Evolution

Christopher Cantwell
ccantwel@usc.edu

Jose Raul Gonzalez Alonso
jrgonzal@usc.edu

USC University of Southern California

Introduction

Quantum evolution has traditionally been a difficult thing to simulate on today's classical computers. This is because the size of the Hilbert space describing a quantum system scales exponentially with the size of the system. As an example, an ensemble of N simple two level systems (qubits) is described by a state vector of length 2^N. It is easy to see that this quickly grows to be unmanageable as we try to explore ever larger systems. At the same time the simulation of such a system is relatively straight forward using either implicit or explicit integration methods. For a system described by a time independent Hamiltonian the simulation can be accomplished very simply by a number of matrix-vector multiplications. There exist a number of BLAS libraries that can accomplish this efficiently. CUBLAS is one such library that performs the multiplication using GPU acceleration. A major bottleneck in GPU computing is communication time thus we would be well served to attempt to fit the entire unitary evolution matrix (U) and state vector (S) into GPU memory and avoid as much communication between the CPU and GPU as possible. For large systems this requires subdividing both U and S into a number of smaller components U_{ij} and S_i. The resulting method of solving for $S' = U * S$ lends itself nicely to parallelization using MPI.

Problem Statement

Matrix-vector multiplication can be recursively decomposed according to the following equation.

$$\left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline U_{21} & U_{22} \end{array}\right)\left(\frac{S_1}{S_2}\right) = \left(\frac{S_1'}{S_2'}\right)$$

We can see that $S_1' = U_{11}S_1 + U_{12}S_2$ and $S_2' = U_{21}S_1 + U_{22}S_2$. This leaves us with the following tasks.

- Solve for U given a Hamiltonian describing a quantum system.
- Determine the number of subdivisions required to fit $U_{i,j}$, S_i, and S'_{ij} into GPU memory.
- Parallelize using MPI.
- Perform matrix-vector multiplication using CUBLAS.

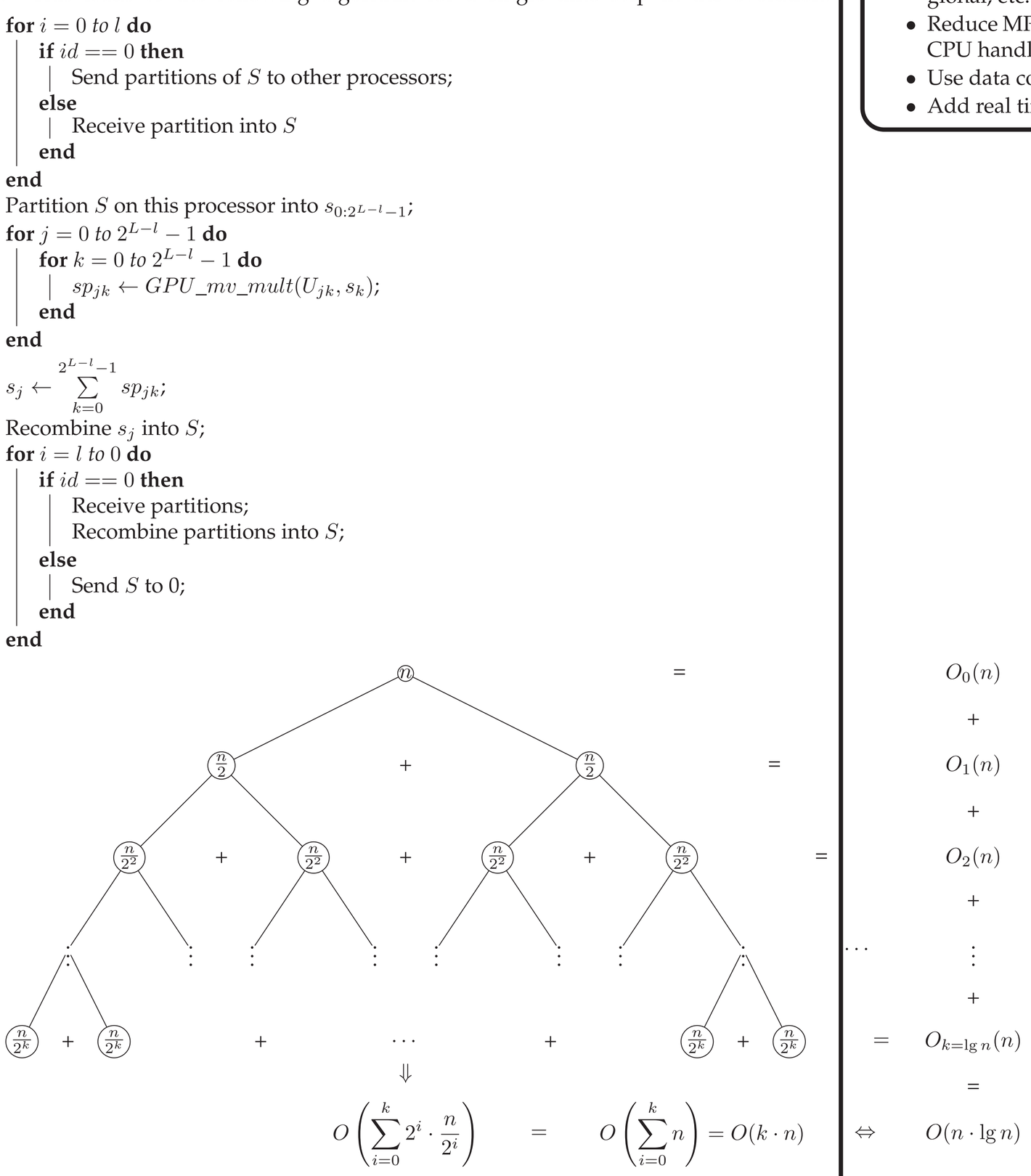
Methodology

The subdivision of U and S leads to a tree structure where each node has four children, one each to compute the four different matrix-vector products. Thus at level l we have 4^l nodes. For a system of N qubits the GPU will need to hold 2^{2N} elements for U_{ij} and 2^{N-L+1} for S_i and S_{ij}. At level k the sub-matrix size will be 2^{2N-k} and the sub-vector 2^{N-k}. The NVIDIA Kepler K20 has a memory size of 5 GB. Assuming we use float precision we solve for the number of levels, L, in the tree we need for the sub-matrix to fit in GPU memory.

$$(2^{2N-L} + 2^{N-L+1})elements \times 2\frac{bytes}{element} \leq 5 \times 10^6 bytes \rightarrow L \geq N - 9$$

Our aim is to simulate large systems so in general 4^{N-9} will be larger than the number of processors we have available. Assuming we have 4^l processors, the remaining $L - l$ subdivisions must be performed on a single processor. This leads to the following algorithm for a single time step in the evolution.

for $i = 0$ *to* l **do**
  if $id == 0$ **then**
    Send partitions of S to other processors;
  else
    Receive partition into S
  end
end
Partition S on this processor into $s_{0:2^{L-l}-1}$;
for $j = 0$ *to* $2^{L-l} - 1$ **do**
  for $k = 0$ *to* $2^{L-l} - 1$ **do**
    $sp_{jk} \leftarrow GPU\_mv\_mult(U_{jk}, s_k)$;
  end
end
$s_j \leftarrow \sum_{k=0}^{2^{L-l}-1} sp_{jk}$;
Recombine s_j into S;
for $i = l$ *to* 0 **do**
  if $id == 0$ **then**
    Receive partitions;
    Recombine partitions into S;
  else
    Send S to 0;
  end
end



Complexity Analysis

To analyze complexity we assume a sequential algorithm using CUBLAS on a single Kepler K20 GPU and a system of N qubits represented by a float precision state vector.

$$T_{seq} \propto 4^{N-9} \times (T_{gpu\_mult} + T_{gpu\_comm})$$

Our MPI parallelized algorithm has the following time complexity

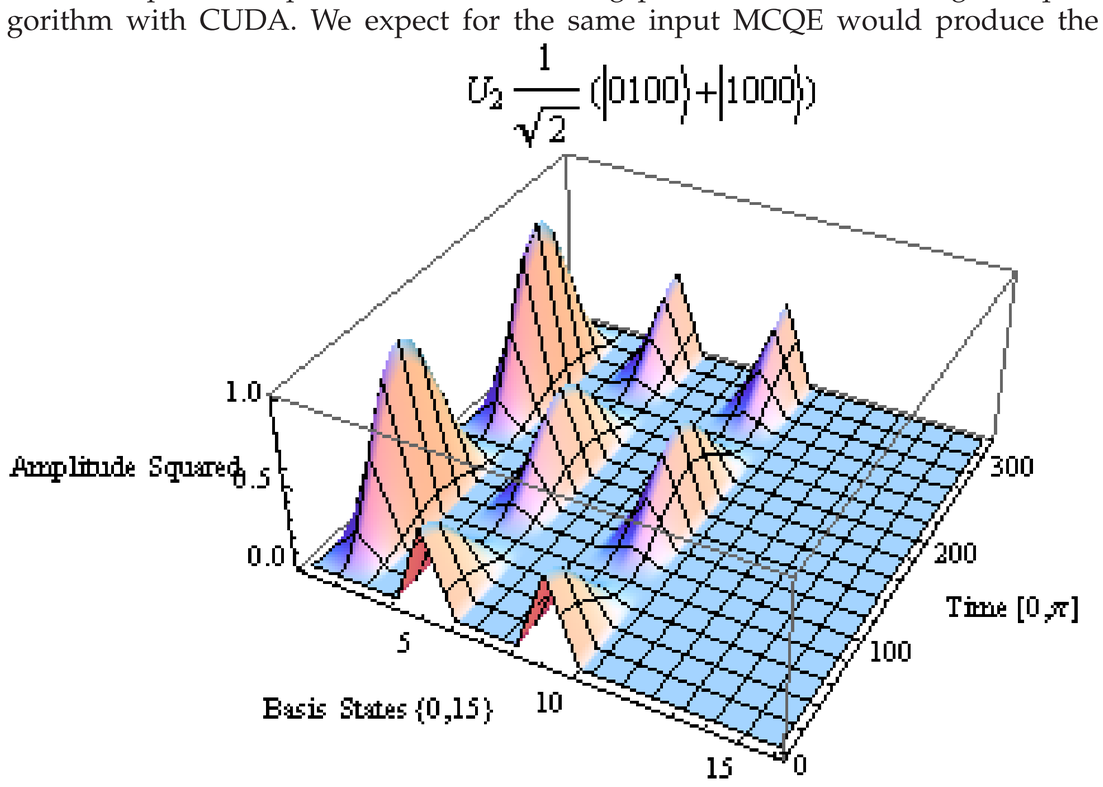
$$T_{MCQE} \propto l \times T_{mpi\_comm} + \frac{2^{2(N-9)-l}}{4^l} \times (T_{gpu\_mult} + T_{gpu\_comm})$$

where 4^l is the number of MPI ranks available to us. Thus our speed up is

$$\frac{l \times T_{mpi\_comm} + 4^{2(N-10-l)} \times (T_{gpu\_mult} + T_{gpu\_comm})}{4^{N-9} \times (T_{gpu\_mult} + T_{gpu\_comm})}$$

Expected Results

Out MCQE algorithm does not use any approximation methods so the result should be exact up to float precision. The following plot was obtained using a sequential algorithm with CUDA. We expect for the same input MCQE would produce the same.



Future Work

There is a lot of room for optimization. The following are possible paths for extension of this program.

- Include optimization for non-dense unitary evolution matrices, i.e. sparse, tridiagional, etc.
- Reduce MPI communication overhead by continuing evolution on the GPU while the CPU handles recombination and partitioning of the sub-vectors.
- Use data compression techniques to reduce memory overhead.
- Add real time rendering of results on the head node.