Title: Virtual Parallel Computing and a Search Algorithm Using Matrix Product States

Date: Apr 20, 2012 02:30 PM

URL: http://pirsa.org/12040079

Abstract: We propose a form of parallel computing on classical computers that is based on matrix product states. The virtual parallelization is accomplished by evolving all possible results for multiple inputs, with bits represented by matrices. The action by classical probabilistic 1-bit and deterministic 2-bit gates such as NAND are implemented in terms of matrix operations and, as opposed to quantum computing, it is possible to copy bits. We present a way to explore this method of computation to solve search problems and count the number of solutions. We argue that if the classical computational cost of testing solutions (witnesses) requires less than O(n^2) local two-bit gates acting on n bits, the search problem can be fully solved in subexponential time. Therefore, for this restricted type of search problem, the virtual parallelization scheme is faster than Groverâ€™s quantum algorithm.

Virtual Parallel Computing with Matrix Product States

Claudio Chamon and Eduardo Mucciolo

BOSTON UNIVERSITY

UCF

NSF

CCF-1116590
CCF-1117241

arXiv:1202.1809

Perimeter Institute, April 20, 2012

Motivation

It is possible to simulate some complex quantum systems efficiently in a classical computer: e.g., spin chains.

Methods that implemented such simulations are based on the Density Matrix Renormalization Group (S. White, 1993):

Motivation

It is possible to simulate some complex quantum systems efficiently in a classical computer: e.g., spin chains.

Methods that implemented such simulations are based on the Density Matrix Renormalization Group (S. White, 1993):

Reasons for success of the method:
- *low spatial dimension*
- *local interactions*
- *gapped spectrum*

Only a small corner of the Hilbert space matters!

Could one use DMRG to mimic a quantum computer?

Motivation

It is possible to simulate some complex quantum systems efficiently in a classical computer: e.g., spin chains.

Methods that implemented such simulations are based on the Density Matrix Renormalization Group (S. White, 1993):

Reasons for success of the method:
- *low spatial dimension*
- *local interactions*
- *gapped spectrum*

Only a small corner of the Hilbert space matters!

Could one use DMRG to mimic a quantum computer?

QC can be done on a line: D. Aharonov, D. Gottesman, S. Irani, J. Kempe (2009)

Random qudit line: R. Movassagh, E. Farhi, J. Goldstone, D. Nagaj, T. J. Osborne, P. W. Shor (prep. 2010)

Matrix Product States

It turns out that one can express many-body wave functions in a form that is suitable for time evolution algorithms that explore decimation or renormalization (G. Vidal 2003):

Matrix Product States

It turns out that one can express many-body wave functions in a form that is suitable for time evolution algorithms that explore decimation or renormalization (G. Vidal 2003):

$$|\Psi\rangle = \sum_{\{x\}} \mathrm{tr}\left[M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}\right] |x_1 x_2 \cdots x_n\rangle$$

two-body, local interactions

$$\hat{U}(t) = e^{-i\hat{H}t} \approx \left(\hat{I} - i\hat{H}\Delta t\right)^N \quad \text{with} \quad \hat{H} = \sum_{i=1}^{n-1} \hat{h}_{i,i+1}$$

⇨ evolution comes down to computing the action of $\hat{h}_{i,i+1}$ on consecutive matrices $M_i^{(x_i)} M_{i+1}^{(x_{i+1})}$.

$$M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \xrightarrow{\hat{h}_{ij}} \mathcal{M}_{ij} \xrightarrow[\text{truncation}]{\text{SVD}} \tilde{M}_i^{(x_i)} \tilde{M}_{i+1}^{(x_{i+1})}$$

small matrix — larger matrix — small matrix

Matrix Product States

It turns out that one can express many-body wave functions in a form that is suitable for time evolution algorithms that explore decimation or renormalization (G. Vidal 2003):

$$|\Psi\rangle = \sum_{\{x\}} \mathrm{tr}\left[M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}\right] |x_1 x_2 \cdots x_n\rangle$$

two-body, local interactions

$$\hat{U}(t) = e^{-i\hat{H}t} \approx \left(\hat{I} - i\hat{H}\Delta t\right)^N \quad \text{with} \quad \hat{H} = \sum_{i=1}^{n-1} \hat{h}_{i,i+1}$$

⇨ evolution comes down to computing the action of $\hat{h}_{i,i+1}$ on consecutive matrices $M_i^{(x_i)} M_{i+1}^{(x_{i+1})}$.

$$M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \xrightarrow{\hat{h}_{ij}} \mathcal{M}_{ij} \xrightarrow[\text{truncation}]{\text{SVD}} \tilde{M}_i^{(x_i)} \tilde{M}_{i+1}^{(x_{i+1})}$$

small matrix — larger matrix — small matrix

Matrix Product States

It turns out that one can express many-body wave functions in a form that is suitable for time evolution algorithms that explore decimation or renormalization (G. Vidal 2003):

$$|\Psi\rangle = \sum_{\{x\}} \mathrm{tr}\left[M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}\right] |x_1 x_2 \cdots x_n\rangle$$

two-body, local interactions

$$\hat{U}(t) = e^{-i\hat{H}t} \approx \left(\hat{I} - i\hat{H}\Delta t\right)^N \quad \text{with} \quad \hat{H} = \sum_{i=1}^{n-1} \hat{h}_{i,i+1}$$

⇨ evolution comes down to computing the action of $\hat{h}_{i,i+1}$ on consecutive matrices $M_i^{(x_i)} M_{i+1}^{(x_{i+1})}$.

$$M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \xrightarrow{\hat{h}_{ij}} \mathcal{M}_{ij} \xrightarrow[\text{truncation}]{\text{SVD}} \tilde{M}_i^{(x_i)} \tilde{M}_{i+1}^{(x_{i+1})}$$

small matrix — larger matrix — small matrix

SVD: singular value decomposition

any $k \times l$ matrix can be decomposed as $M = U \Lambda V$

$[U]_{k\times k}$ $[V]_{l\times l}$ *unitary matrices*

$$[\Lambda]_{k\times l} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \dots & \\ & & & \lambda_D \end{pmatrix}$$ *positive diagonal matrix*

$D = \max\{k, l\}$

$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$ *singular values*

SVD: singular value decomposition

any $k \times l$ matrix can be decomposed as $M = U \Lambda V$

$[U]_{k\times k}$ $[V]_{l\times l}$ *unitary matrices*

$$[\Lambda]_{k\times l} = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \dots & \\ & & & \lambda_D \end{pmatrix}$$ *positive diagonal matrix*

$D = \max\{k, l\}$

$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_D \geq 0$ *singular values*

Question

- What is it that gives quantum computer its power?

Question

- What is it that gives quantum computer its power?

  - parallelism? kind of... ... measurement spoils it most of the times

Question

- What is it that gives quantum computer its power?

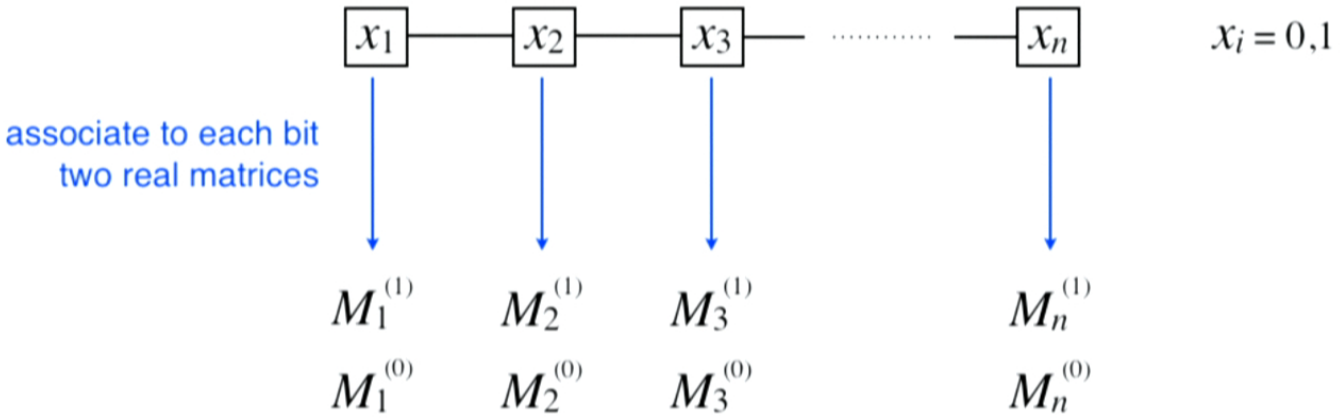
  - parallelism? kind of... ... measurement spoils it most of the times

- Can we find inspiration from quantum systems to improve classical algorithms?

  - parallelism?

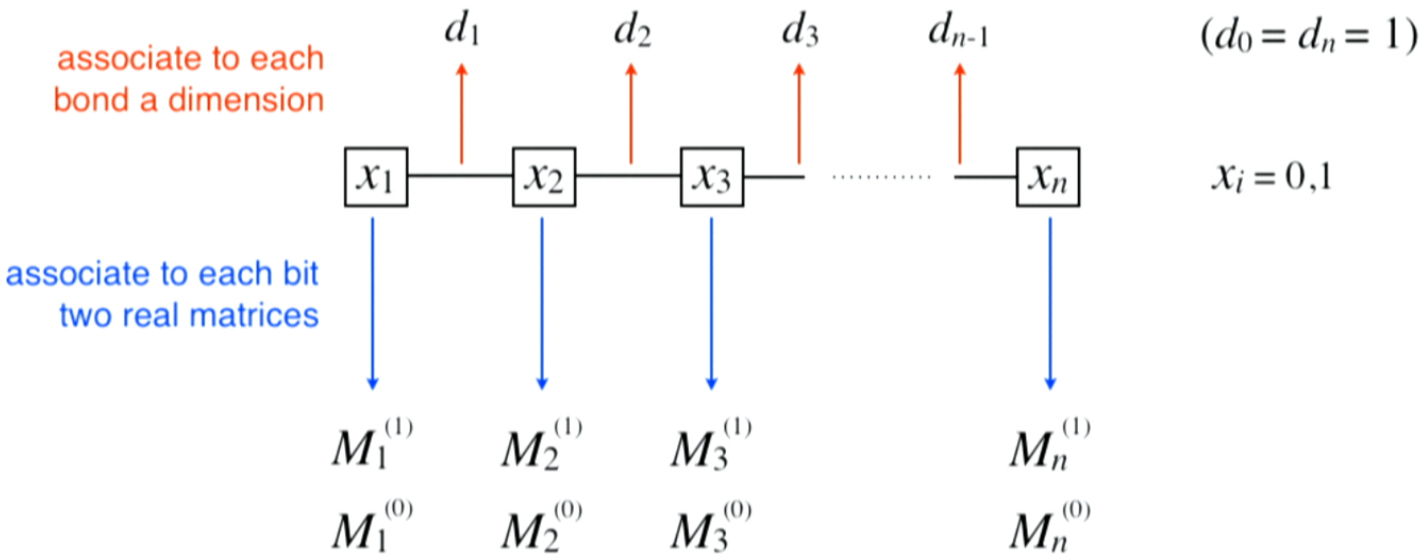
Our Method

Consider a string of *n* bits:



Our Method

Consider a string of *n* bits:



matrix dimensions:

$$\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i}$$

edge bit matrices are vectors:

$$\left[M_1^{(x_1)}\right]_{1\times d_2}$$

$$\left[M_n^{(x_n)}\right]_{d_{n-1}\times 1}$$

The probability distribution is normalized:

$$P(x_1, x_2, \ldots, x_n) = \frac{1}{Z} M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}$$

The probability distribution is normalized:

$$P(x_1, x_2, \ldots, x_n) = \frac{1}{Z} M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}$$

The probability distribution is normalized:

$$P(x_1, x_2, \ldots, x_n) = \frac{1}{Z} M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}$$

$$Z = \sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}$$

$$= \left(M_1^{(0)} + M_1^{(1)}\right) \left(M_2^{(0)} + M_2^{(1)}\right) \cdots \left(M_n^{(0)} + M_n^{(1)}\right)$$

normalization constant

The probability distribution is normalized:

$$P(x_1, x_2, \ldots, x_n) = \frac{1}{Z} M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}$$

$$Z = \sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} M_1^{(x_1)} M_2^{(x_2)} \cdots M_n^{(x_n)}$$

$$= \left(M_1^{(0)} + M_1^{(1)}\right)\left(M_2^{(0)} + M_2^{(1)}\right) \cdots \left(M_n^{(0)} + M_n^{(1)}\right)$$

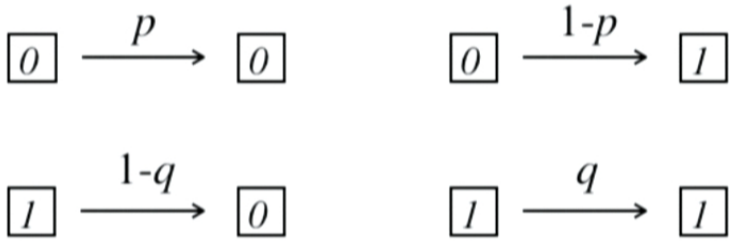
normalization constant

$$\Rightarrow \quad \sum_{x_1=0,1} \sum_{x_2=0,1} \cdots \sum_{x_n=0,1} P(x_1, x_2, \ldots, x_n) = 1$$

- As gates are applied to the bit string, $P(x_1,x_2,\ldots,x_n)$ evolves.

- Gates preserve the normalization of the probability distribution.

One-bit gates: *probabilistic*



transfer matrix

$$t^{0,0} = p$$
$$t^{0,1} = 1 - p$$
$$t^{1,0} = 1 - q$$
$$t^{1,1} = q$$

One-bit gates: *probabilistic*

$0 \xrightarrow{p} 0$ $\qquad$ $0 \xrightarrow{1\text{-}p} 1$

$1 \xrightarrow{1\text{-}q} 0$ $\qquad$ $1 \xrightarrow{q} 1$

transfer matrix

$$t^{0,0} = p$$
$$t^{0,1} = 1 - p$$
$$t^{1,0} = 1 - q$$
$$t^{1,1} = q$$

⇒ changes the bit matrices

$$\tilde{M}_i^{(x_i)} = \sum_{x_i'=0,1} t^{x_i,x_i'} M_i^{(x_i')}$$

Examples:

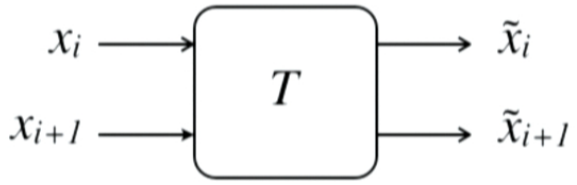
NOT gate $(p = q = 0)$

$$\tilde{M}_i^{(0)} = M_i^{(1)}$$
$$\tilde{M}_i^{(1)} = M_i^{(0)}$$

RAND gate $(p = q = 1/2)$

$$\tilde{M}_i^{(0)} = \tilde{M}_i^{(1)} = \frac{1}{2}\left(M_i^{(0)} + M_i^{(1)}\right)$$

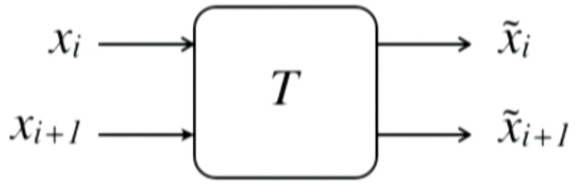
Two-bit gates: *deterministic*



defined by two logical functions $A(a,b)$, $B(a,b)$

transfer matrix $T^{\tilde{a}\tilde{b},ab} = \begin{cases} 1, & \tilde{a} = A(a,b) \text{ and } \tilde{b} = B(a,b) \\ 0, & \text{otherwise} \end{cases}$

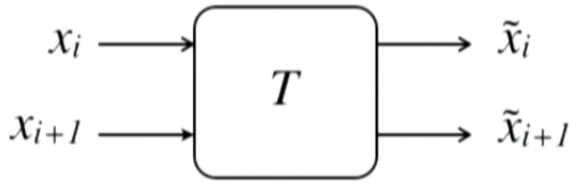
Two-bit gates: *deterministic*



defined by two logical functions $A(a,b)$, $B(a,b)$

transfer matrix $T^{\tilde{a}\tilde{b},ab} = \begin{cases} 1, & \tilde{a} = A(a,b) \text{ and } \tilde{b} = B(a,b) \\ 0, & \text{otherwise} \end{cases}$

Two-bit gates: *deterministic*



defined by two logical functions $A(a,b)$, $B(a,b)$

transfer matrix $$T^{\tilde{a}\tilde{b},ab} = \begin{cases} 1, & \tilde{a} = A(a,b) \text{ and } \tilde{b} = B(a,b) \\ 0, & \text{otherwise} \end{cases}$$

- The new bit matrices must satisfy

$$\tilde{M}_i^{(x_i)}\, \tilde{M}_{i+1}^{(x_{i+1})} = \sum_{x'_i, x'_{i+1}=0,1} T^{x_i x_{i+1}, x'_i, x'_{i+1}}\, M_i^{(x'_i)}\, M_{i+1}^{(x'_{i+1})}$$

Example: NAND gate

$$A_{\mathrm{NAND}}(a,b) = a$$

$$B_{\mathrm{NAND}}(a,b) = \overline{a \wedge b}$$

Example: NAND gate

$$A_{\mathrm{NAND}}(a,b) = a$$

$$B_{\mathrm{NAND}}(a,b) = \overline{a \wedge b}$$

$$T^{01,00} = T^{01,01} = T^{11,10} = T^{10,11} = 1$$

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

Example: NAND gate

$$A_{\mathrm{NAND}}(a,b) = a$$

$$B_{\mathrm{NAND}}(a,b) = \overline{a \wedge b}$$

$$\Rightarrow \quad T^{01,00} = T^{01,01} = T^{11,10} = T^{10,11} = 1$$

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

composed matrix

$$\mathcal{M}^{\mathrm{NAND}}_{i,i+1} = \left(\begin{array}{c|c} 0 & M_i^{(0)} M_{i+1}^{(0)} + M_i^{(0)} M_{i+1}^{(1)} \\ \hline M_i^{(1)} M_{i+1}^{(1)} & M_i^{(1)} M_{i+1}^{(0)} \end{array}\right)$$

$$\overset{\mathrm{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

$$\left[M_i^{(x_i)}\right]_{d_{i-1} \times d_i} \quad \left[M_{i+1}^{(x_{i+1})}\right]_{d_i \times d_{i+1}}$$

Example: NAND gate

$$A_{\mathrm{NAND}}(a,b) = a$$

$$B_{\mathrm{NAND}}(a,b) = \overline{a \wedge b}$$

$$\Rightarrow \quad T^{01,00} = T^{01,01} = T^{11,10} = T^{10,11} = 1$$

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

composed matrix

$$\mathcal{M}^{\mathrm{NAND}}_{i,i+1} = \left(\begin{array}{c|c} 0 & M_i^{(0)} M_{i+1}^{(0)} + M_i^{(0)} M_{i+1}^{(1)} \\ \hline M_i^{(1)} M_{i+1}^{(1)} & M_i^{(1)} M_{i+1}^{(0)} \end{array}\right)$$

$$\overset{\mathrm{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

$$\left[M_i^{(x_i)}\right]_{d_{i-1} \times d_i} \left[M_{i+1}^{(x_{i+1})}\right]_{d_i \times d_{i+1}} \longrightarrow \left[\tilde{M}_i^{(x_i)}\right]_{d_{i-1} \times \tilde{d}_i} \left[\tilde{M}_{i+1}^{(x_{i+1})}\right]_{\tilde{d}_i \times d_{i+1}}$$

Example: NAND gate

$$A_{\mathrm{NAND}}(a,b) = a$$

$$B_{\mathrm{NAND}}(a,b) = \overline{a \wedge b}$$

$$\Rightarrow \quad T^{01,00} = T^{01,01} = T^{11,10} = T^{10,11} = 1$$

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

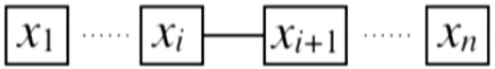
composed matrix

$$\mathcal{M}^{\mathrm{NAND}}_{i,i+1} = \left(\begin{array}{c|c} 0 & M_i^{(0)} M_{i+1}^{(0)} + M_i^{(0)} M_{i+1}^{(1)} \\ \hline M_i^{(1)} M_{i+1}^{(1)} & M_i^{(1)} M_{i+1}^{(0)} \end{array}\right)$$

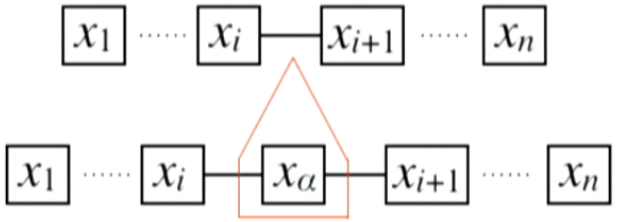
$$\overset{\mathrm{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

$$\left[M_i^{(x_i)}\right]_{d_{i-1} \times \boxed{d_i}} \left[M_{i+1}^{(x_{i+1})}\right]_{\boxed{d_i} \times d_{i+1}} \longrightarrow \left[\tilde{M}_i^{(x_i)}\right]_{d_{i-1} \times \boxed{\tilde{d}_i}} \left[\tilde{M}_{i+1}^{(x_{i+1})}\right]_{\boxed{\tilde{d}_i} \times d_{i+1}}$$

Bit insertion:



Bit insertion:



$$M_1^{(x_1)} \cdots M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

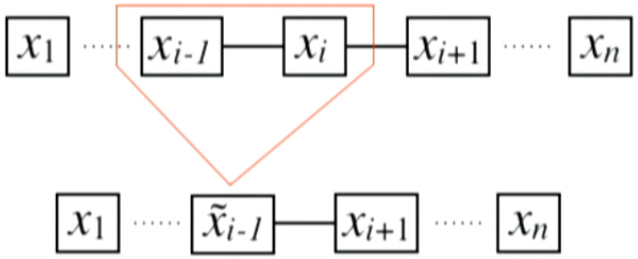
$$M_1^{(x_1)} \cdots M_i^{(x_i)} M_\alpha^{(x_\alpha)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

$$\left[M_\alpha^{(x_\alpha)} \right]_{d_i \times d_i}$$

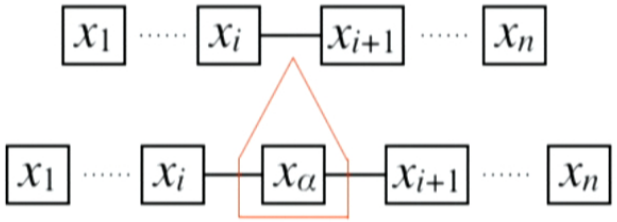
$$M_\alpha^{(0)} = I \quad \text{identity matrix}$$

$$M_\alpha^{(1)} = 0 \quad \text{null matrix}$$

Bit erasure:



Bit insertion:



$$M_1^{(x_1)} \cdots M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

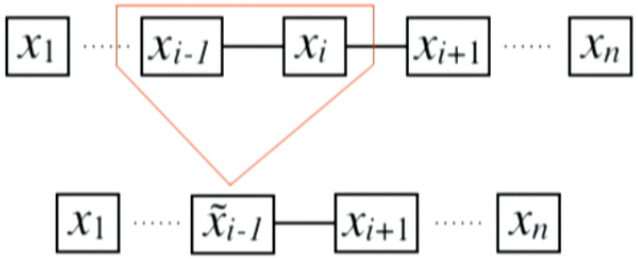
$$M_1^{(x_1)} \cdots M_i^{(x_i)} M_\alpha^{(x_\alpha)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

$$\left[M_\alpha^{(x_\alpha)} \right]_{d_i \times d_i}$$

$$M_\alpha^{(0)} = I \quad \text{identity matrix}$$

$$M_\alpha^{(1)} = 0 \quad \text{null matrix}$$

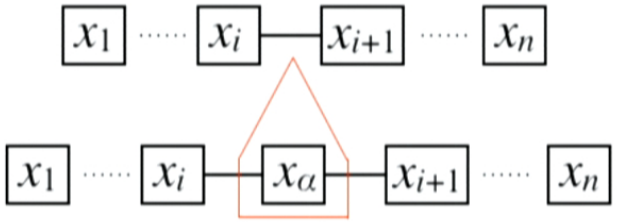
Bit erasure:



$$M_1^{(x_1)} \cdots M_{i-1}^{(x_{i-1})} M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

$$M_1^{(x_1)} \cdots \tilde{M}_{i-1}^{(x_{i-1})} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

Bit insertion:



$$M_1^{(x_1)} \cdots M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

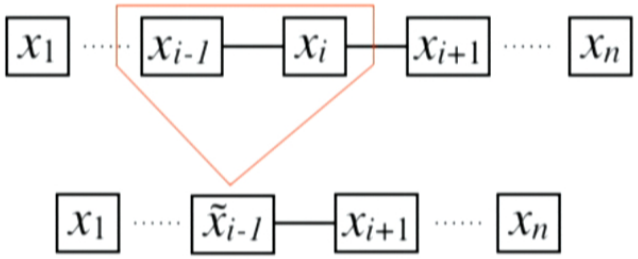
$$M_1^{(x_1)} \cdots M_i^{(x_i)} M_\alpha^{(x_\alpha)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

$$\left[M_\alpha^{(x_\alpha)} \right]_{d_i \times d_i}$$

$$M_\alpha^{(0)} = I \quad \text{identity matrix}$$

$$M_\alpha^{(1)} = 0 \quad \text{null matrix}$$

Bit erasure:



$$M_1^{(x_1)} \cdots M_{i-1}^{(x_{i-1})} M_i^{(x_i)} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

$$M_1^{(x_1)} \cdots \tilde{M}_{i-1}^{(x_{i-1})} M_{i+1}^{(x_{i+1})} \cdots M_n^{(x_n)}$$

$$\tilde{M}_{i-1}^{(x_{i-1})} = M_{i-1}^{(x_{i-1})} \left(M_i^{(0)} + M_i^{(1)} \right)$$

Search Algorithm based on Matrix Product States

Consider a function $y = f(x)$ which can be implemented with $\mathcal{O}(n^p)$ gates.

Given a certain y^* we want to find x^* such that $y^* = f(x^*)$.

Standard approach: for every value of x, compute $f(x)$ until you get y^*.

MPS approach:
- create a superposition of all values of x
- apply all gates defining $f(x)$
- evaluate $P(y^*)$
- find $P(y^*) = \frac{m}{2^n}$, where m is the number of solutions
- modify the input x bit by bit and repeat the calculation

Search Algorithm based on Matrix Product States

Consider a function $y = f(x)$ which can be implemented with $\mathcal{O}(n^p)$ gates.

Given a certain y^* we want to find x^* such that $y^* = f(x^*)$.

Standard approach: for every value of x, compute $f(x)$ until you get y^*.

MPS approach:
- create a superposition of all values of x
- apply all gates defining $f(x)$
- evaluate $P(y^*)$
- find $P(y^*) = \frac{m}{2^n}$, where m is the number of solutions
- modify the input x bit by bit and repeat the calculation

Search Algorithm based on Matrix Product States

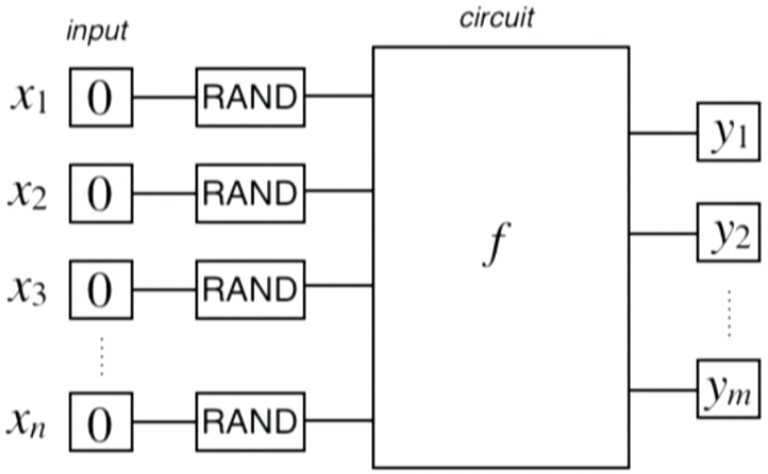
Consider a function $y = f(x)$ which can be implemented with $\mathcal{O}(n^p)$ gates.

Given a certain y^* we want to find x^* such that $y^* = f(x^*)$.

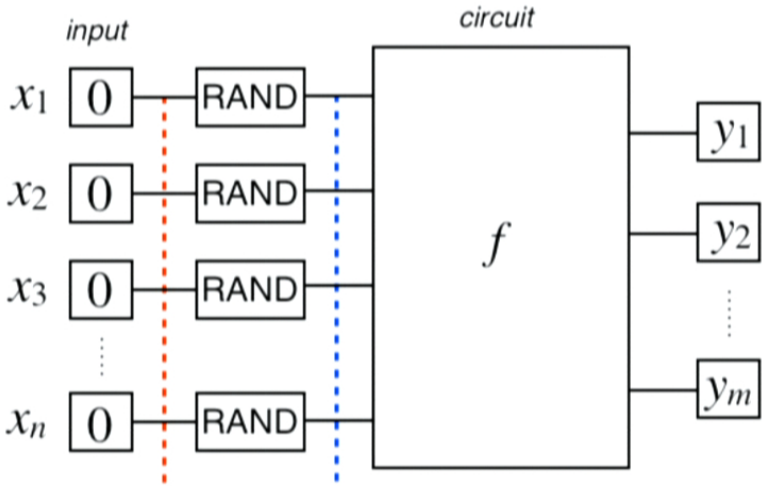
Standard approach: for every value of x, compute $f(x)$ until you get y^*.

MPS approach:
- create a superposition of all values of x
- apply all gates defining $f(x)$
- evaluate $P(y^*)$
- find $P(y^*) = \frac{m}{2^n}$, where m is the number of solutions
- modify the input x bit by bit and repeat the calculation

Example:



Example:



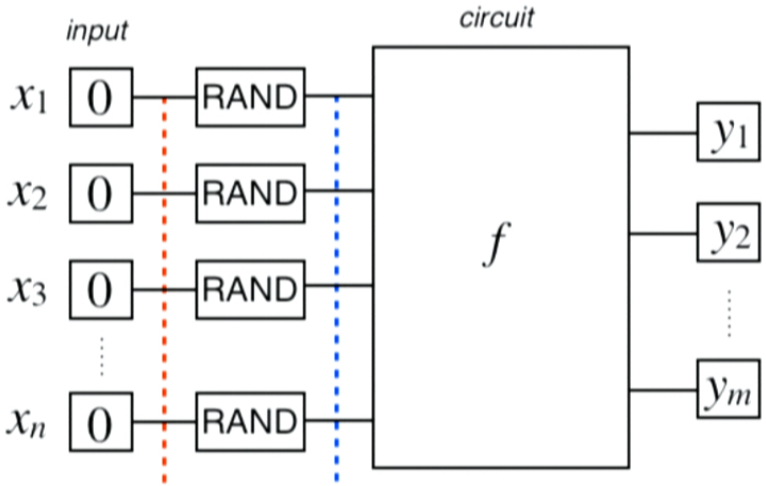
$$|\Psi\rangle = |0\rangle_1 |0\rangle_2 |0\rangle_3 \cdots |0\rangle_n$$

$$M_i^{(0)} = 1$$
$$M_i^{(1)} = 0$$

$$|\Psi\rangle = \frac{1}{2^n} \left(|0\rangle + |1\rangle\right)_1 \left(|0\rangle + |1\rangle\right)_2 \left(|0\rangle + |1\rangle\right)_3 \cdots \left(|0\rangle + |1\rangle\right)_n$$

$$M_i^{(0)} = M_i^{(1)} = \frac{1}{2}$$

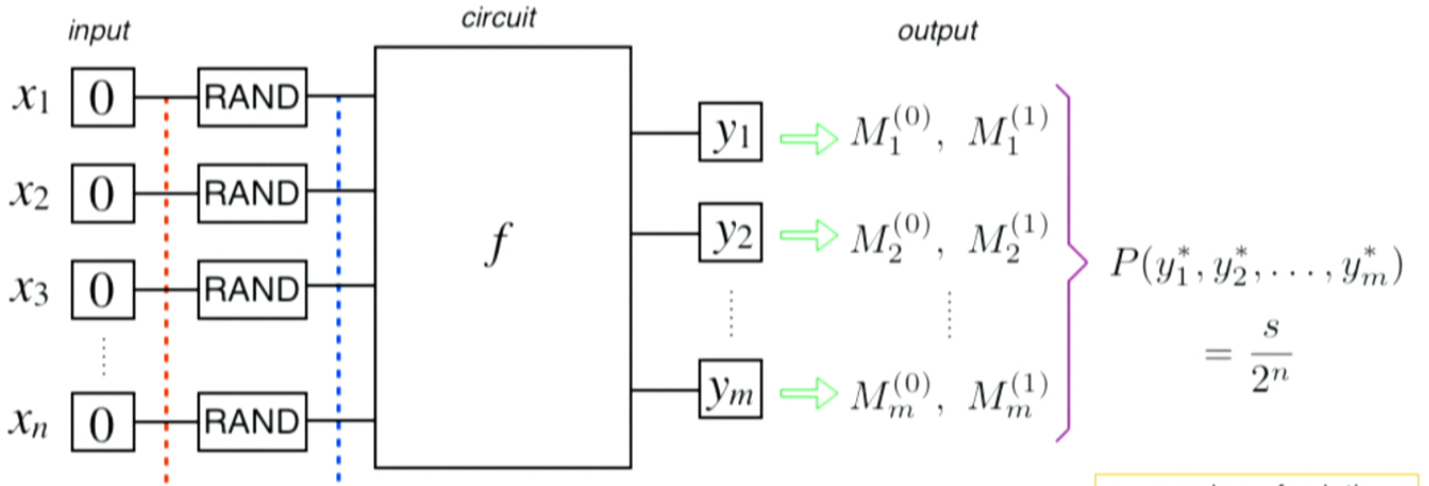
Example:



$$|\Psi\rangle = |0\rangle_1 |0\rangle_2 |0\rangle_3 \cdots |0\rangle_n \qquad M_i^{(0)} = 1 \quad M_i^{(1)} = 0$$

$$|\Psi\rangle = \frac{1}{2^n} (|0\rangle + |1\rangle)_1 (|0\rangle + |1\rangle)_2 (|0\rangle + |1\rangle)_3 \cdots (|0\rangle + |1\rangle)_n \qquad M_i^{(0)} = M_i^{(1)} = \frac{1}{2}$$

Example:



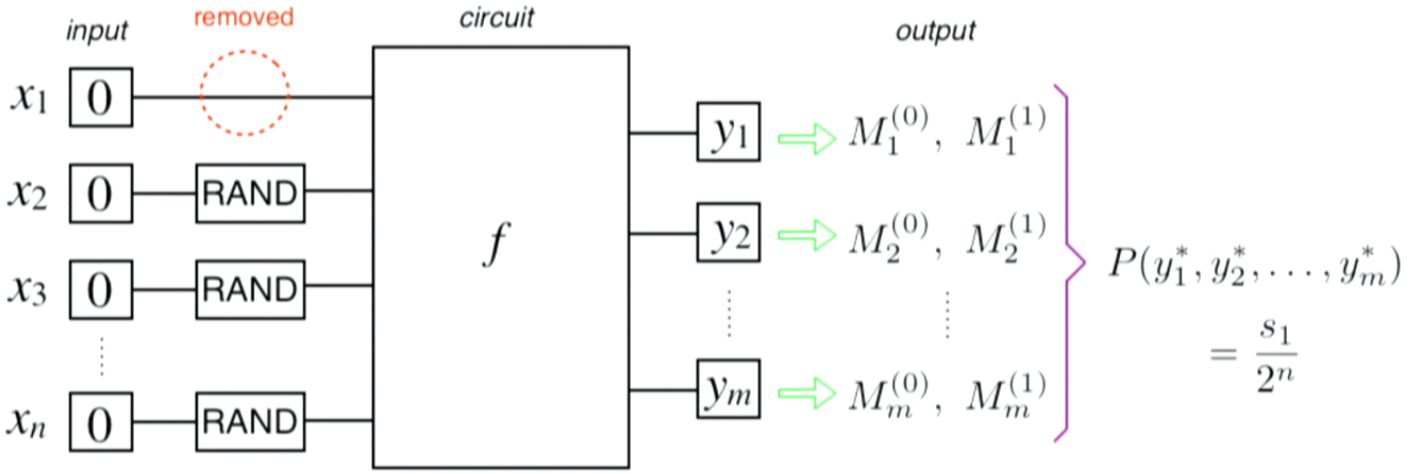
$$|\Psi\rangle = |0\rangle_1 |0\rangle_2 |0\rangle_3 \cdots |0\rangle_n$$

$$M_i^{(0)} = 1$$
$$M_i^{(1)} = 0$$

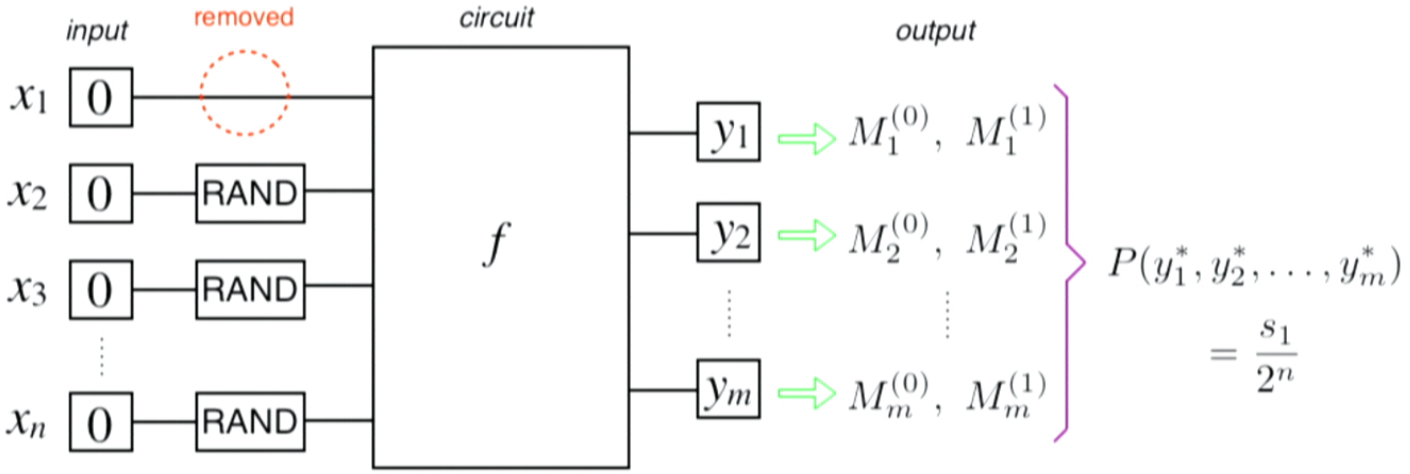
$$|\Psi\rangle = \frac{1}{2^n} (|0\rangle + |1\rangle)_1 (|0\rangle + |1\rangle)_2 (|0\rangle + |1\rangle)_3 \cdots (|0\rangle + |1\rangle)_n$$

$$M_i^{(0)} = M_i^{(1)} = \frac{1}{2}$$

Example (cont.):



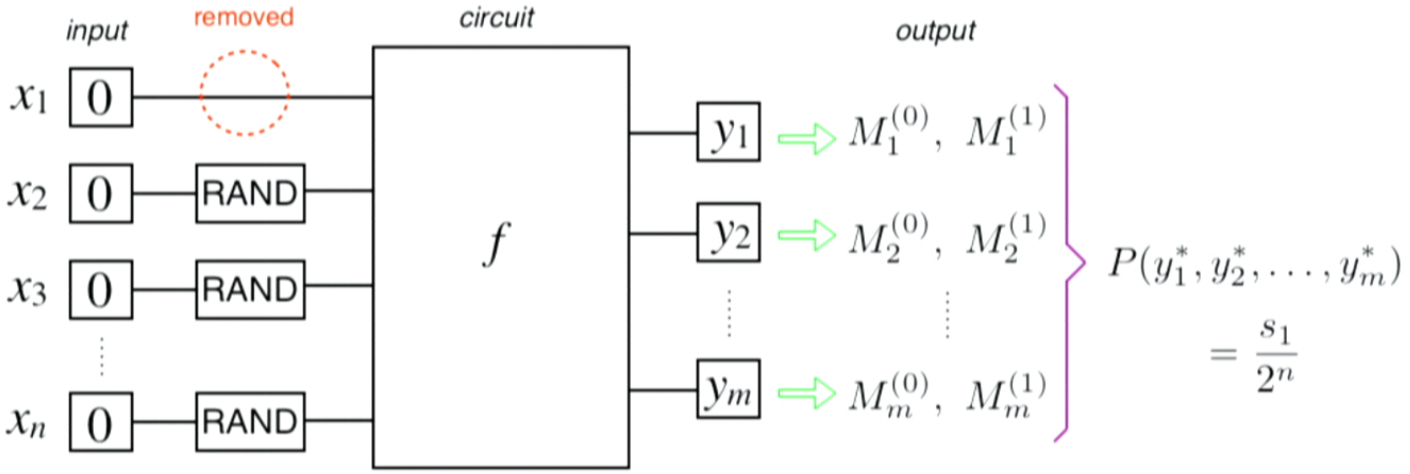
Example (cont.):



$s_1 \geq 1 \Rightarrow$ keep $x_1 = 0$

$s_1 < 1 \Rightarrow$ set $x_1 = 1$

Example (cont.):



$s_1 \geq 1 \Rightarrow$ keep $x_1 = 0$

$s_1 < 1 \Rightarrow$ set $x_1 = 1$

Repeat this procedure for all remaining n-1 bits

and find $x_1^*, x_2^*, \ldots, x_n^*$

Computational Cost

SVD $\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i}$ $\left[M_{i+1}^{(x_{i+1})}\right]_{d_i\times d_{i+1}}$ $\sim \mathcal{O}\left(d_i^3\right)$

Computational Cost

SVD $\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i}$ $\left[M_{i+1}^{(x_{i+1})}\right]_{d_i\times d_{i+1}}$ $\sim \mathcal{O}\left(d_i^3\right)$

n_g = number of two-bit gates

D = maximum bond dimension

⇨ circuit computational cost $\sim \mathcal{O}\left(n_g \times D^3\right)$

need to reapply the circuit n times

⇨ algorithm computational cost $\sim \mathcal{O}\left(n \times n_g \times D^3\right)$

Computational Cost

SVD $\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i}$ $\left[M_{i+1}^{(x_{i+1})}\right]_{d_i\times d_{i+1}}$ $\sim \mathcal{O}\left(d_i^3\right)$

n_g = number of two-bit gates

D = maximum bond dimension

⇒ circuit computational cost $\sim \mathcal{O}\left(n_g \times D^3\right)$

need to reapply the circuit n times

⇒ algorithm computational cost $\sim \mathcal{O}\left(n \times n_g \times D^3\right)$

Computational Cost

SVD $\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i}$ $\left[M_{i+1}^{(x_{i+1})}\right]_{d_i\times d_{i+1}}$ $\sim \mathcal{O}\left(d_i^3\right)$

n_g = number of two-bit gates

D = maximum bond dimension

⇨ circuit computational cost $\sim \mathcal{O}\left(n_g \times D^3\right)$

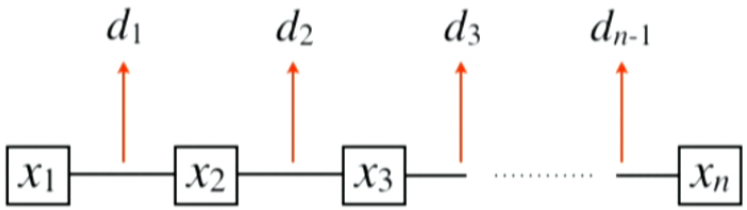
need to reapply the circuit n times

⇨ algorithm computational cost $\sim \mathcal{O}\left(n \times n_g \times D^3\right)$

Given $f(x)$, how does D scale with n_g and n?

$$D \leq D_{\max} = \min\left(2^{\lfloor\sqrt{2n_g}\rfloor}, 2^{\lfloor n/2\rfloor}\right)$$

Proof:



$(d_0 = d_n = 1)$

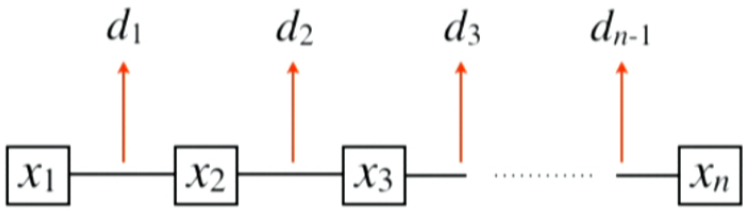
$x_i = 0,1$

need to determine the growth of bond dimensions with the number of 2-bit gates

Given $f(x)$, how does D scale with n_g and n?

$$D \leq D_{\max} = \min\left(2^{\lfloor\sqrt{2n_g}\rfloor}, 2^{\lfloor n/2\rfloor}\right)$$

Proof:



$(d_0 = d_n = 1)$

$x_i = 0,1$

need to determine the growth of bond dimensions with the number of 2-bit gates

Action of 2-bit gates

Recall, e.g., the NAND gate

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

composed matrix

$$\mathcal{M}^{\mathrm{NAND}}_{i,i+1} = \left(\begin{array}{c:c} 0 & M_i^{(0)}M_{i+1}^{(0)} + M_i^{(0)}M_{i+1}^{(1)} \\ \hdashline M_i^{(1)}M_{i+1}^{(1)} & M_i^{(1)}M_{i+1}^{(0)} \end{array}\right) \overset{\mathrm{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

Action of 2-bit gates

Recall, e.g., the NAND gate

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

composed matrix

$$\mathcal{M}^{\mathrm{NAND}}_{i,i+1} = \left(\begin{array}{c:c} 0 & M_i^{(0)} M_{i+1}^{(0)} + M_i^{(0)} M_{i+1}^{(1)} \\ \hdashline M_i^{(1)} M_{i+1}^{(1)} & M_i^{(1)} M_{i+1}^{(0)} \end{array}\right) \overset{\mathrm{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

$$\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i} \quad \left[M_{i+1}^{(x_{i+1})}\right]_{d_i \times d_{i+1}} \longrightarrow \left[\tilde{M}_i^{(x_i)}\right]_{d_{i-1}\times \tilde{d}_i} \quad \left[\tilde{M}_{i+1}^{(x_{i+1})}\right]_{\tilde{d}_i \times d_{i+1}}$$

Action of 2-bit gates

Recall, e.g., the NAND gate

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

composed matrix

$$\mathcal{M}^{\text{NAND}}_{i,i+1} = \left(\begin{array}{c:c} 0 & M_i^{(0)}M_{i+1}^{(0)} + M_i^{(0)}M_{i+1}^{(1)} \\ \hdashline M_i^{(1)}M_{i+1}^{(1)} & M_i^{(1)}M_{i+1}^{(0)} \end{array}\right) \overset{\text{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

$$\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i} \left[M_{i+1}^{(x_{i+1})}\right]_{d_i\times d_{i+1}} \longrightarrow \left[\tilde{M}_i^{(x_i)}\right]_{d_{i-1}\times \tilde{d}_i} \left[\tilde{M}_{i+1}^{(x_{i+1})}\right]_{\tilde{d}_i\times d_{i+1}}$$

$$\left[\mathcal{M}^{\text{gate}}_{i,i+1}\right]_{2d_{i-1}\times 2d_{i+1}} \overset{\text{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix}_{2d_{i-1}\times \tilde{d}_i} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}_{\tilde{d}_i\times 2d_{i+1}}$$

$$\tilde{d}_i = \min(2d_{i-1}, 2d_{i+1})$$

Action of 2-bit gates

Recall, e.g., the NAND gate

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

composed matrix

$$\mathcal{M}^{\text{NAND}}_{i,i+1} = \left(\begin{array}{c:c} 0 & M_i^{(0)}M_{i+1}^{(0)} + M_i^{(0)}M_{i+1}^{(1)} \\ \hdashline M_i^{(1)}M_{i+1}^{(1)} & M_i^{(1)}M_{i+1}^{(0)} \end{array}\right) \overset{\text{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}$$

$$\left[M_i^{(x_i)}\right]_{d_{i-1}\times d_i} \left[M_{i+1}^{(x_{i+1})}\right]_{d_i\times d_{i+1}} \longrightarrow \left[\tilde{M}_i^{(x_i)}\right]_{d_{i-1}\times \tilde{d}_i} \left[\tilde{M}_{i+1}^{(x_{i+1})}\right]_{\tilde{d}_i\times d_{i+1}}$$

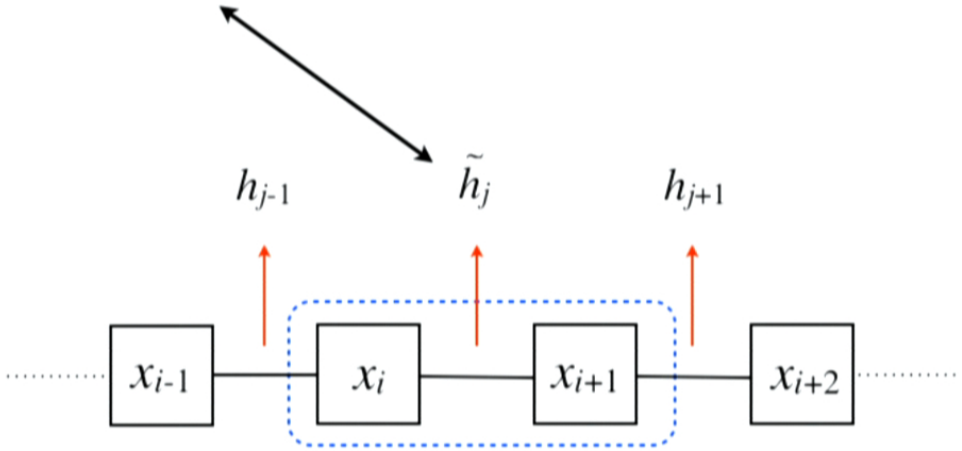
$$\left[\mathcal{M}^{\text{gate}}_{i,i+1}\right]_{2d_{i-1}\times 2d_{i+1}} \overset{\text{SVD}}{=} \begin{pmatrix} \tilde{M}_i^{(0)} \\ \\ \tilde{M}_i^{(1)} \end{pmatrix}_{2d_{i-1}\times \tilde{d}_i} \begin{pmatrix} \tilde{M}_{i-1}^{(0)} & \tilde{M}_{i-1}^{(1)} \end{pmatrix}_{\tilde{d}_i\times 2d_{i+1}}$$

$$\tilde{d}_i = \min(2d_{i-1}, 2d_{i+1})$$

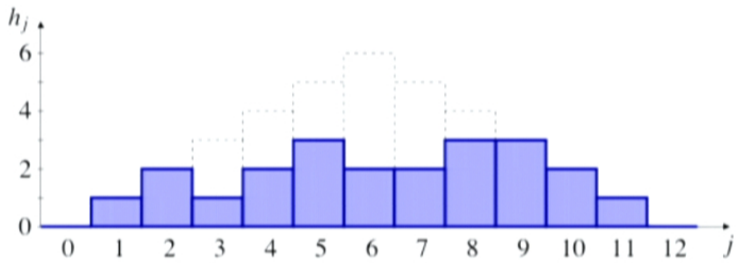
Action of 2-bit gates

$\tilde{d}_i = \min(2d_{i-1}, 2d_{i+1})$ define "heights" $\tilde{h}_i = \log_2 d_i$

$\tilde{h}_i = \min(h_{i-1}, h_{i+1}) + 1$



Evolution of heights with computational steps



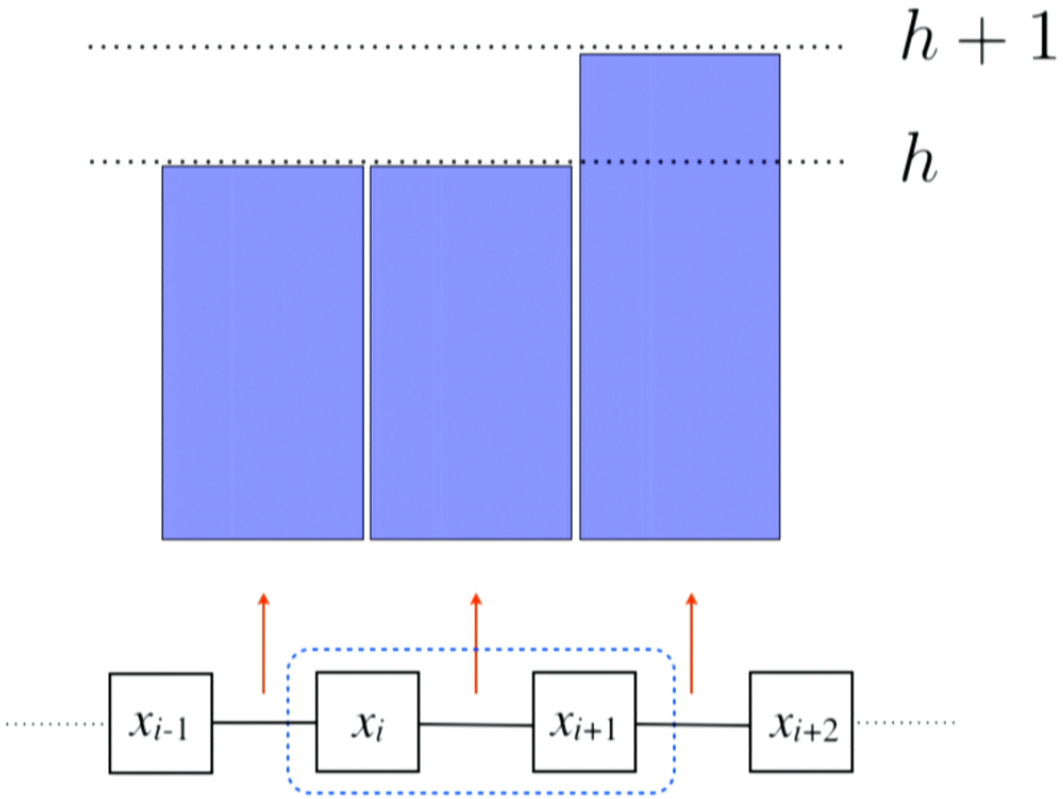
height profile satisfies $|h_i - h_{i-1}| \leq 1, \forall i$

proof by induction:

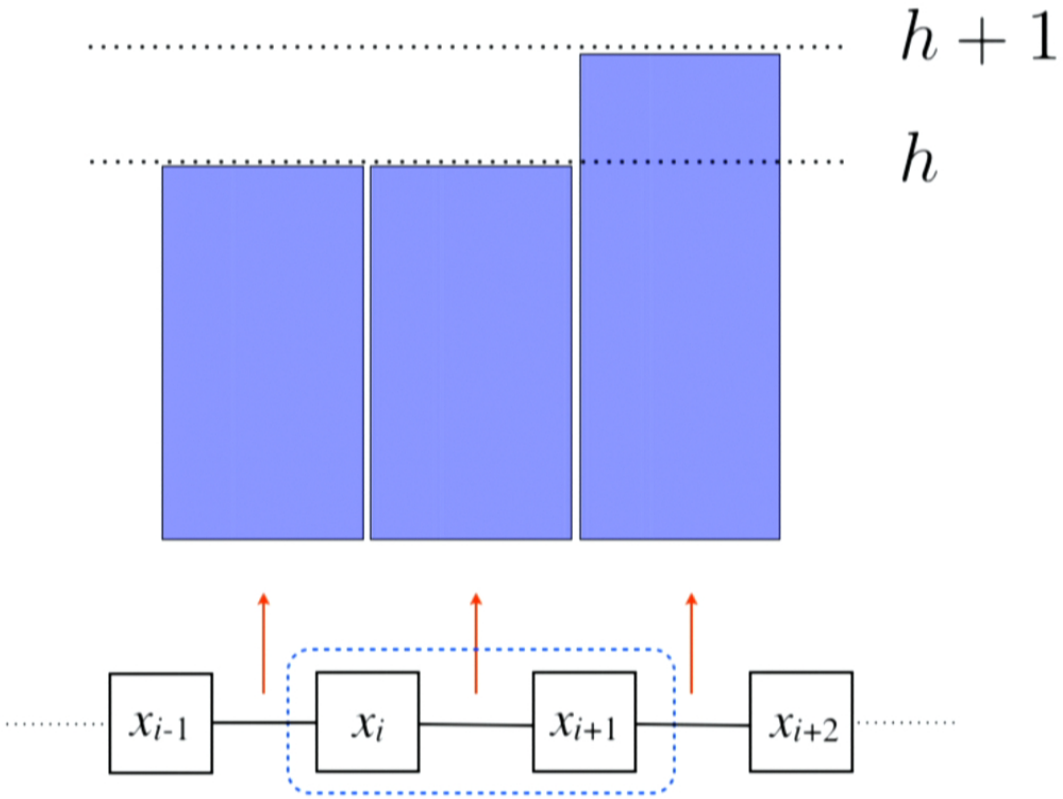
initial input is a product state $h_i = 0 \quad \forall i$

if condition is satisfied at step τ , it is satisfied at step $\tau + 1$

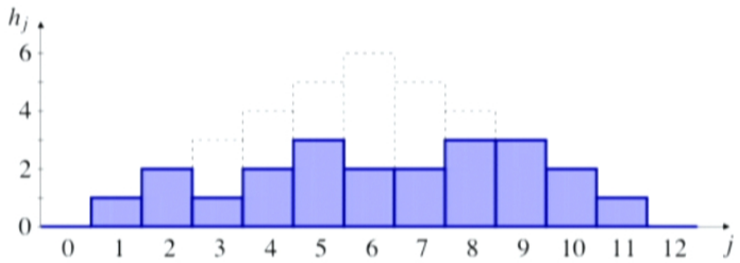
e.g. (1 of 9 cases)



e.g. (1 of 9 cases)



Evolution of heights with computational steps



height profile satisfies $|h_i - h_{i-1}| \leq 1, \forall i$

proof by induction:

initial input is a product state $h_i = 0 \quad \forall i$

if condition is satisfied at step τ , it is satisfied at step $\tau + 1$

comment: bit insertions and bit removals

$$\Delta h_i = h_i(\tau + 1) - h_i(\tau) \leq 2$$

$$\Delta h_i = h_i(\tau + 1) - h_i(\tau) \leq 2$$

area satisfies $$S = \sum_i h_i \leq 2\, n_g$$

$$\Delta h_i = h_i(\tau + 1) - h_i(\tau) \leq 2$$

area satisfies $$S = \sum_i h_i \leq 2\, n_g$$

height profile satisfies $$\left| h_i - h_{i-1} \right| \leq 1, \forall i$$

$$\Rightarrow \quad h_{\max} \leq \min\left(\lfloor \sqrt{2\, n_g} \rfloor, \lfloor n/2 \rfloor \right)$$

$$h_{\max} \leq \min\left(\lfloor\sqrt{2\,n_g}\rfloor, \lfloor n/2\rfloor\right)$$

"uniform case" $n_g \sim n \times \tau \;\rightarrow \tau \sim n \;\; (\text{or } L)$

to reach max "entanglement"

A. Hamma, S. Santra, and P. Zanardi, arXiv:1109.4391

Given $f(x)$, how does D scale with n_g and n?

$$D \leq D_{\max} = \min\left(2^{\lfloor\sqrt{2n_g}\rfloor}, 2^{\lfloor n/2\rfloor}\right)$$

Suppose $n_g \sim \mathcal{O}(n^p)$. We have two scenarios:

(i) if $p < 2 \implies D_{\max} \sim \mathcal{O}\left(2^{\sqrt{2}\, n^{p/2}}\right)$

algorithm cost $\sim \mathcal{O}\left(n^{p+1} \times 2^{3\sqrt{2}\, n^{p/2}}\right)$ *subexponential*

Critical point: $p_c = 2$

When $n_g < \mathcal{O}\left(n^2\right)$, the MPS search algorithm beats Grover's.

$\mathcal{O}\left(2^{n/2}\right)$

Critical point: $p_c = 2$

When $n_g < \mathcal{O}\left(n^2\right)$, the MPS search algorithm beats Grover's. $\mathcal{O}\left(2^{n/2}\right)$

catch: Grover's algorithm works for any unsorted data base, whereas the requirement $n_g < \mathcal{O}\left(n^2\right)$ may only be applicable to structured data base (we do not know for sure).

Critical point: $p_c = 2$

When $n_g < \mathcal{O}\left(n^2\right)$, the MPS search algorithm beats Grover's. $\mathcal{O}\left(2^{n/2}\right)$

catch: Grover's algorithm works for any unsorted data base, whereas the requirement $n_g < \mathcal{O}\left(n^2\right)$ may only be applicable to structured data base (we do not know for sure).

Critical point: $p_c = 2$

When $n_g < \mathcal{O}\left(n^2\right)$, the MPS search algorithm beats Grover's. $\mathcal{O}\left(2^{n/2}\right)$

catch: Grover's algorithm works for any unsorted data base, whereas the requirement $n_g < \mathcal{O}\left(n^2\right)$ may only be applicable to structured data base (we do not know for sure).

Critical point: $p_c = 2$

When $n_g < \mathcal{O}\left(n^2\right)$, the MPS search algorithm beats Grover's. $\mathcal{O}\left(2^{n/2}\right)$

catch: Grover's algorithm works for any unsorted data base, whereas the requirement $n_g < \mathcal{O}\left(n^2\right)$ may only be applicable to structured data base (we do not know for sure).

But we can use the same method to solve other computational problems as well:

- satisfiability (2-SAT and #2-SAT) (in progress)

A topic for discussion

- Is there a “physical” interpretation to bond dimension growth?

A topic for discussion

- Is there a “physical” interpretation to bond dimension growth?

Complexity from physics perspective

truth table

x_i	x_{i+1}	$\tilde{x}_i$	$\tilde{x}_{i+1}$
0	0	0	1
1	0	1	1
0	1	0	1
1	1	1	0

Complexity from physics perspective

truth table

x_i	x_{i+1}	$\tilde{x}_{i+1}$
0	0	1
1	0	1
0	1	1
1	1	0

$$\tilde{M}^{(0)}_{i+1} = M^{(1)}_i M^{(1)}_{i+1}$$
$$\tilde{M}^{(1)}_{i+1} = M^{(0)}_i M^{(0)}_{i+1} + M^{(0)}_i M^{(1)}_{i+1} + M^{(1)}_i M^{(0)}_{i+1}$$

and remove bit i

Complexity from physics perspective

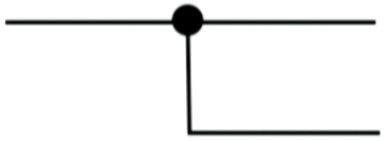
truth table

x_i	x_{i+1}	$\tilde{x}_{i+1}$
0	0	1
1	0	1
0	1	1
1	1	0

$$\tilde{M}^{(0)}_{i+1} = M^{(1)}_i M^{(1)}_{i+1}$$
$$\tilde{M}^{(1)}_{i+1} = M^{(0)}_i M^{(0)}_{i+1} + M^{(0)}_i M^{(1)}_{i+1} + M^{(1)}_i M^{(0)}_{i+1}$$

and remove bit i

Complexity from physics perspective

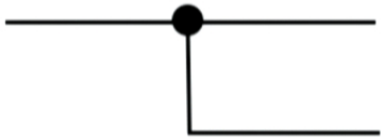


FANOUT
insert bit + copy gate

wire cross
SWAP gate

ONLY 2bit gates needed!

Complexity from physics perspective

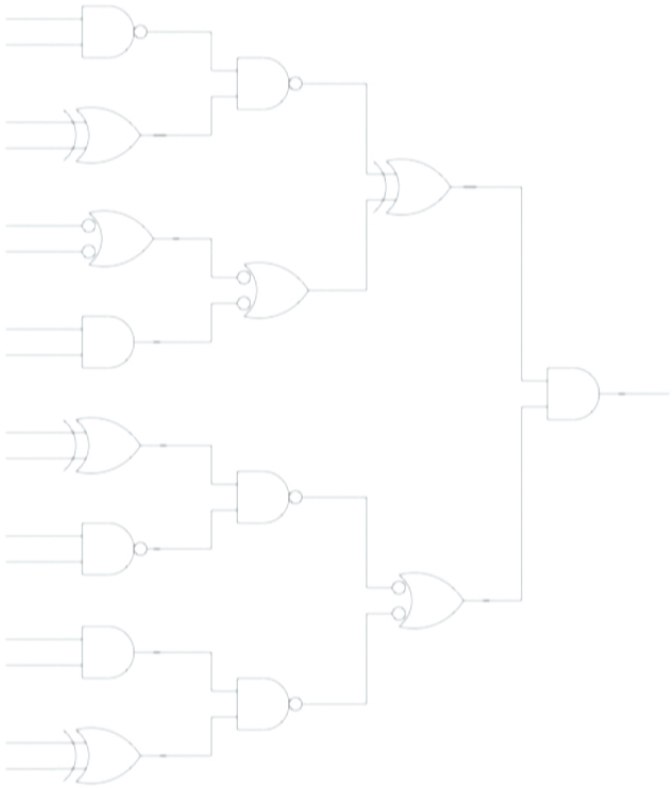


What is the physical measure of circuit complexity?

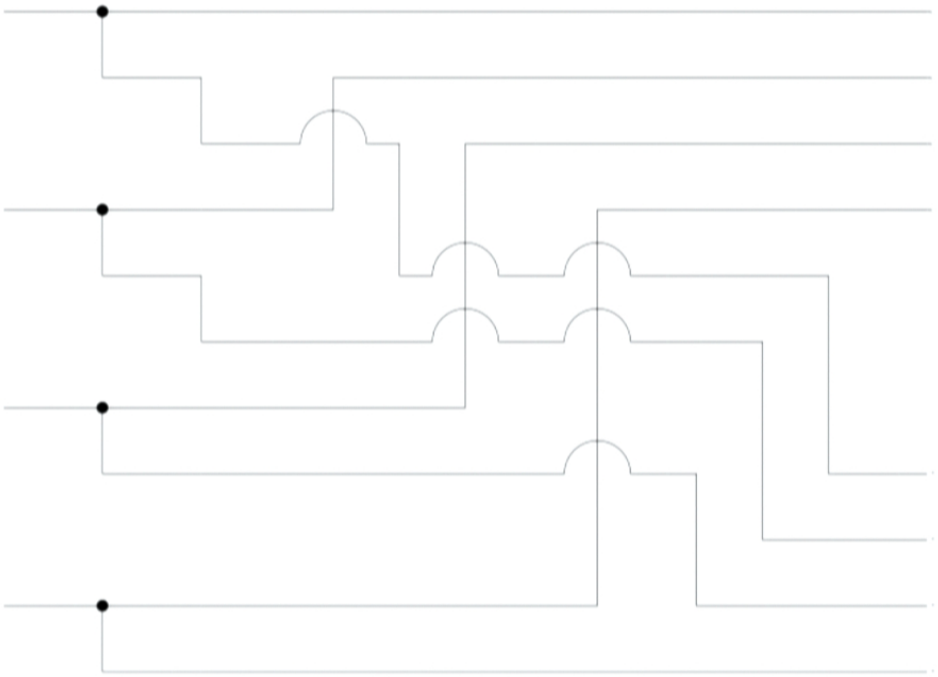
Entanglement!

LITERALLY!!

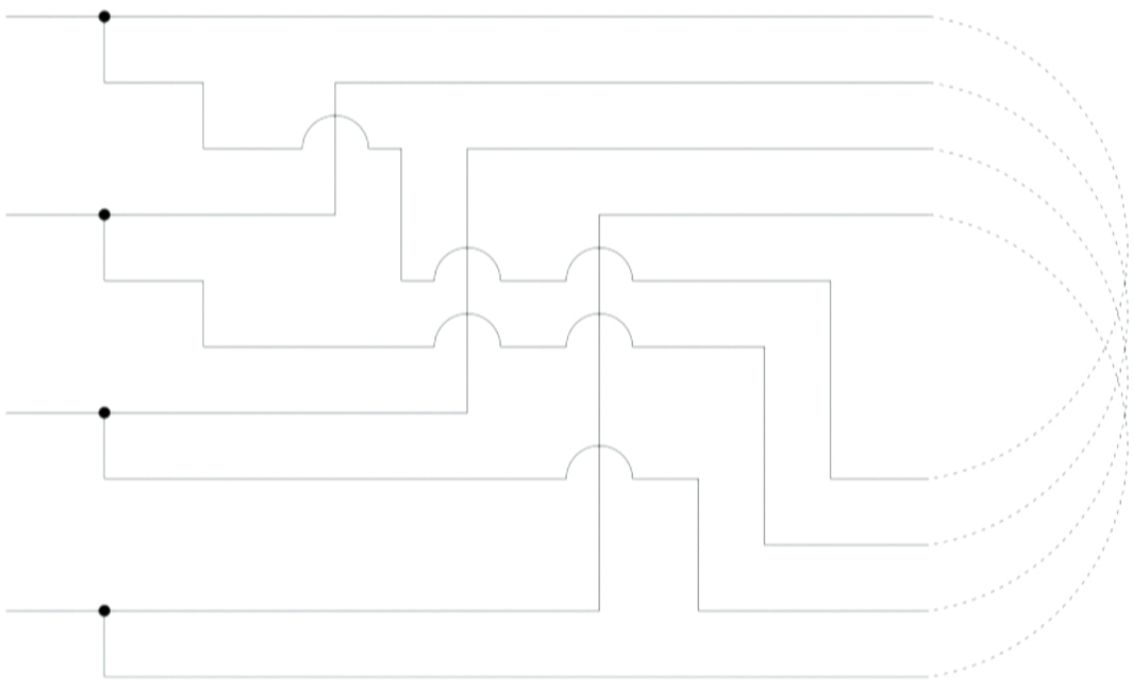
A trivial case with no entanglement



A case with a **LOT** of entanglement



A case with a **LOT** of entanglement



Some topics for discussion

Some topics for discussion

- Do all two-bit gates increase the bond dimension?

Some topics for discussion

- Do all two-bit gates increase the bond dimension?

- Can we use truncation of singular values to limit bond dimension growth?

Some topics for discussion

- Do all two-bit gates increase the bond dimension?

- Can we use truncation of singular values to limit bond dimension growth?

- Can one use matrix computing to do number factoring?

Some topics for discussion

- Do all two-bit gates increase the bond dimension?

- Can we use truncation of singular values to limit bond dimension growth?

- Can one use matrix computing to do number factoring?

Some topics for discussion

- Do all two-bit gates increase the bond dimension?

- Can we use truncation of singular values to limit bond dimension growth?

- Can one use matrix computing to do number factoring?

- Have we tested the method in practice?

Some topics for discussion

- Do all two-bit gates increase the bond dimension?

- Can we use truncation of singular values to limit bond dimension growth?

- Can one use matrix computing to do number factoring?

- Have we tested the method in practice?

THE END