# Remedies Towards Breaching The Memory Wall for Sparse Computations

Mert Hidayetoglu[1], Leopold Grinberg[2], Constantinos Evangelinos[2], Jinjun Xiong[2], and Wen-mei W. Hwu[1]

[1]University of Illinois at Urbana-Champaign

[2]IBM T. J. Watson Research Center

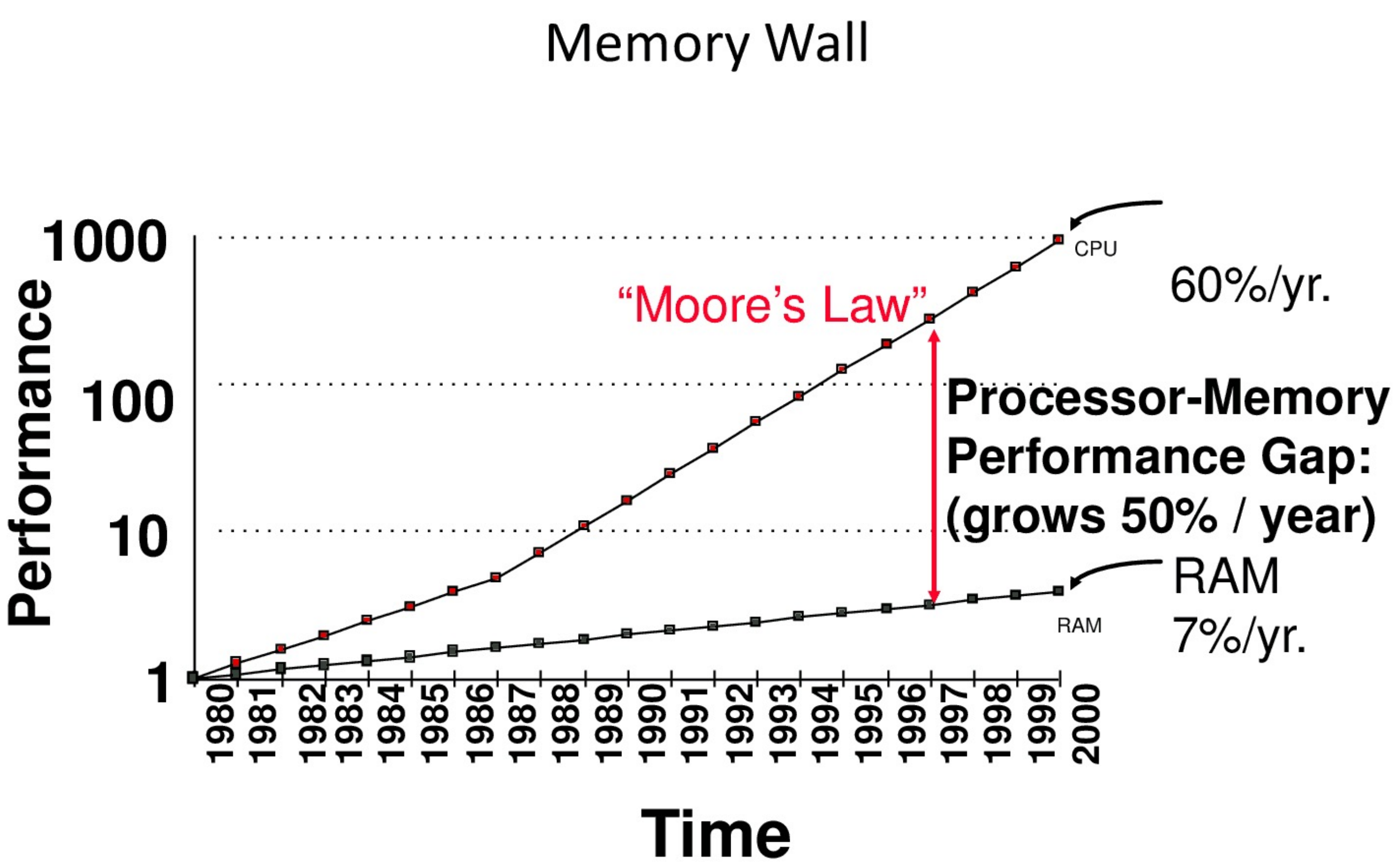
ECE ILLINOIS Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

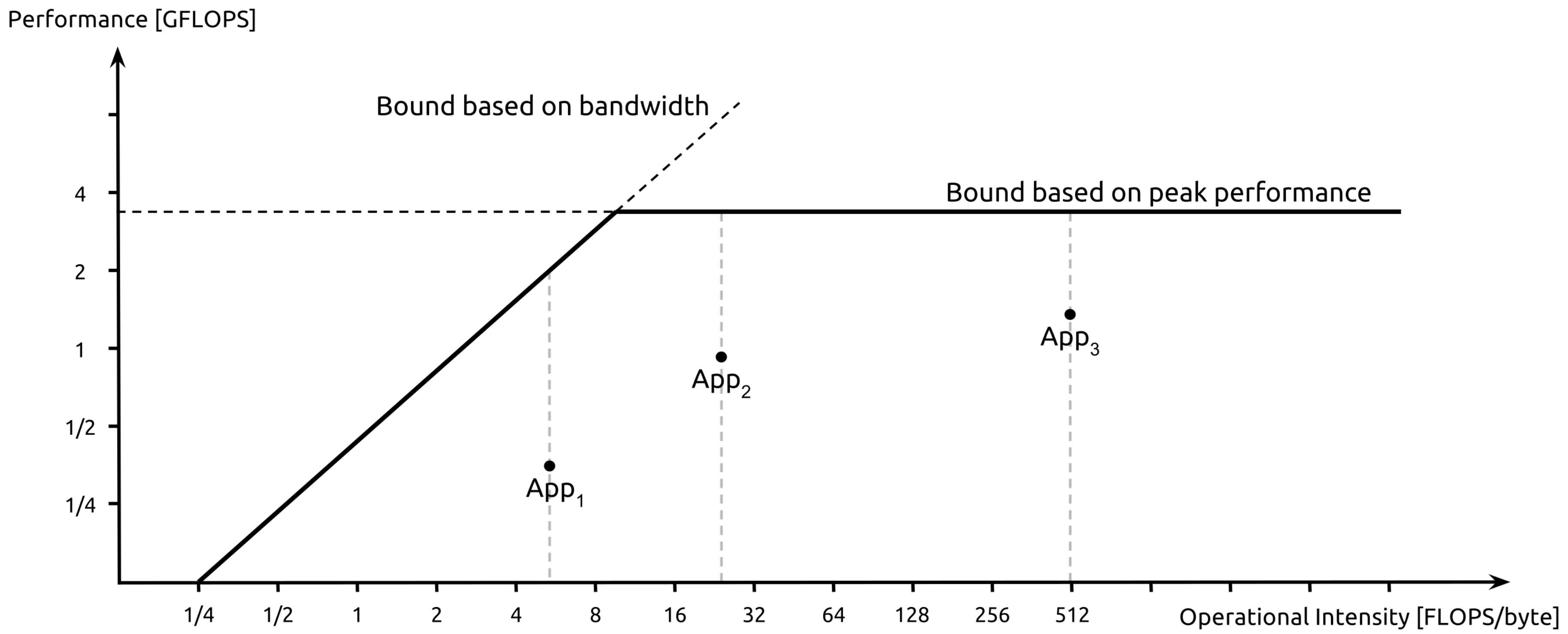
C3SR center for cognitive computing systems research

# Memory Wall

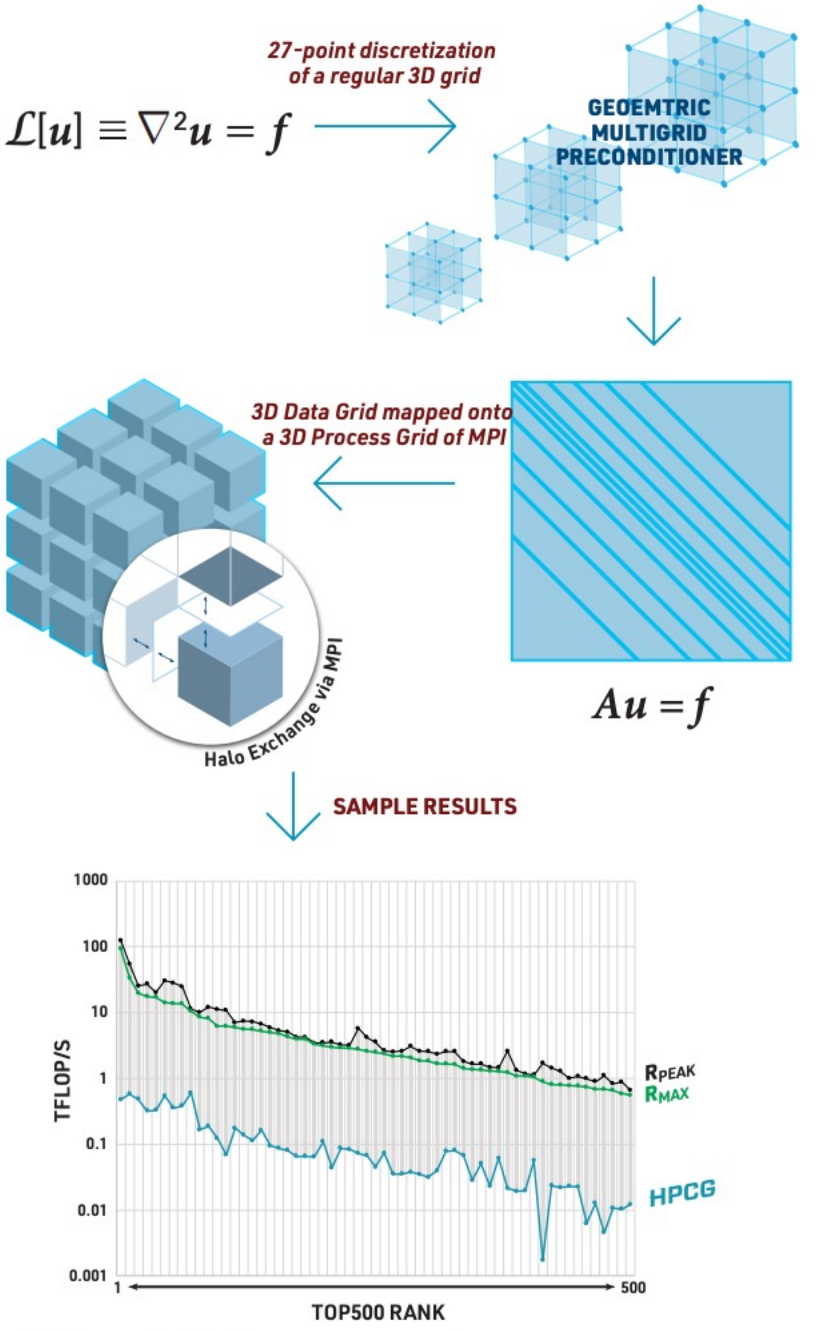


Memory Wall
Performance
1000
100
10
1
CPU
60%/yr.
"Moore's Law"
Processor-Memory
Performance Gap:
(grows 50% / year)
RAM
7%/yr.
RAM
1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000
Time

# Roofline Model




```
//SPMV
for(int m = 0; m < nump; m++){
  double reduce = 0;
  for(int n = displ[m]; n < displ[m+1]; n++)
    reduce += value[n]*x[index[n]];
  y[m] = reduce;
}
```




www.hpcg-benchmark.org

# HPCG

HPCG List for June 2019

| Rank | TOP500 Rank | System | Cores | Rmax (TFlop/s) | HPCG (TFlop/s) |
|---|---|---|---|---|---|
| 1 | 1 | **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 2,414,592 | 148,600.0 | 2925.75 |
| 2 | 2 | **Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox<br>DOE/NNSA/LLNL<br>United States | 1,572,480 | 94,640.0 | 1795.67 |
| 3 | 20 | K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect , Fujitsu<br>RIKEN Advanced Institute for Computational Science (AICS)<br>Japan | 705,024 | 10,510.0 | 602.74 |
| 4 | 7 | **Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Intel Xeon Phi 7250 68C 1.4GHz, Aries interconnect , Cray Inc.<br>DOE/NNSA/LANL/SNL<br>United States | 979,072 | 20,158.7 | 546.12 |
| 5 | 8 | **AI Bridging Cloud Infrastructure (ABCI)** - PRIMERGY CX2570 M4, Xeon Gold 6148 20C 2.4GHz, NVIDIA Tesla V100 SXM2, Infiniband EDR , Fujitsu<br>National Institute of Advanced Industrial Science and Technology (AIST)<br>Japan | 391,680 | 19,880.0 | 508.85 |

www.top500.org

**Sparse data**

**Low FLOPS**

**Stresses memory syst.**

$\mathcal{O}(N)$ Memory

$\mathcal{O}(N)$ Comput.

# LINPACK

| Rank | System | Cores | Rmax (TFlop/s) | Rpeak (TFlop/s) | Power (kW) |
|---|---|---|---|---|---|
| 1 | **Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM<br>DOE/SC/Oak Ridge National Laboratory<br>United States | 2,414,592 | 148,600.0 | 200,794.9 | 10,096 |
| 2 | **Sierra** - IBM Power System S922LC, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband , IBM / NVIDIA / Mellanox<br>DOE/NNSA/LLNL<br>United States | 1,572,480 | 94,640.0 | 125,712.0 | 7,438 |
| 3 | **Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway , NRCPC<br>National Supercomputing Center in Wuxi<br>China | 10,649,600 | 93,014.6 | 125,435.9 | 15,371 |
| 4 | **Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000 , NUDT<br>National Super Computer Center in Guangzhou<br>China | 4,981,760 | 61,444.5 | 100,678.7 | 18,482 |
| 5 | **Frontera** - Dell C6420, Xeon Platinum 8280 28C 2.7GHz, Mellanox InfiniBand HDR , Dell EMC<br>Texas Advanced Computing Center/Univ. of Texas<br>United States | 448,448 | 23,516.4 | 38,745.9 | |

www.top500.org

**Dense data**

**High FLOPS**

**Stresses processors**

$\mathcal{O}(N^2)$ Memory

$\mathcal{O}(N^3)$ Comput.

# Computational Kernels in Each Iteration

**Other HPCG Operations**

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int m = 0; m < nump; m++){
  double reduce = b[m];
  for(int n = displ[m]; n < displ[m+1]; n++)
    reduce -= value[n]*x[index[n]];
  x[m] += reduce/diag[m];
}
```

Inherently Sequential

**Other HPCG Operations**

```
//GAUSS-SEIDEL BACKWARD SWEEP
for(int m = nump-1; m > -1; m--){
  double reduce = b[m];
  for(int n = displ[m+1]-1; n > displ[m]-1; n--)
    reduce -= value[n]*x[index[n]];
  x[m] += reduce/diag[m];
}
```

Inherently Sequential

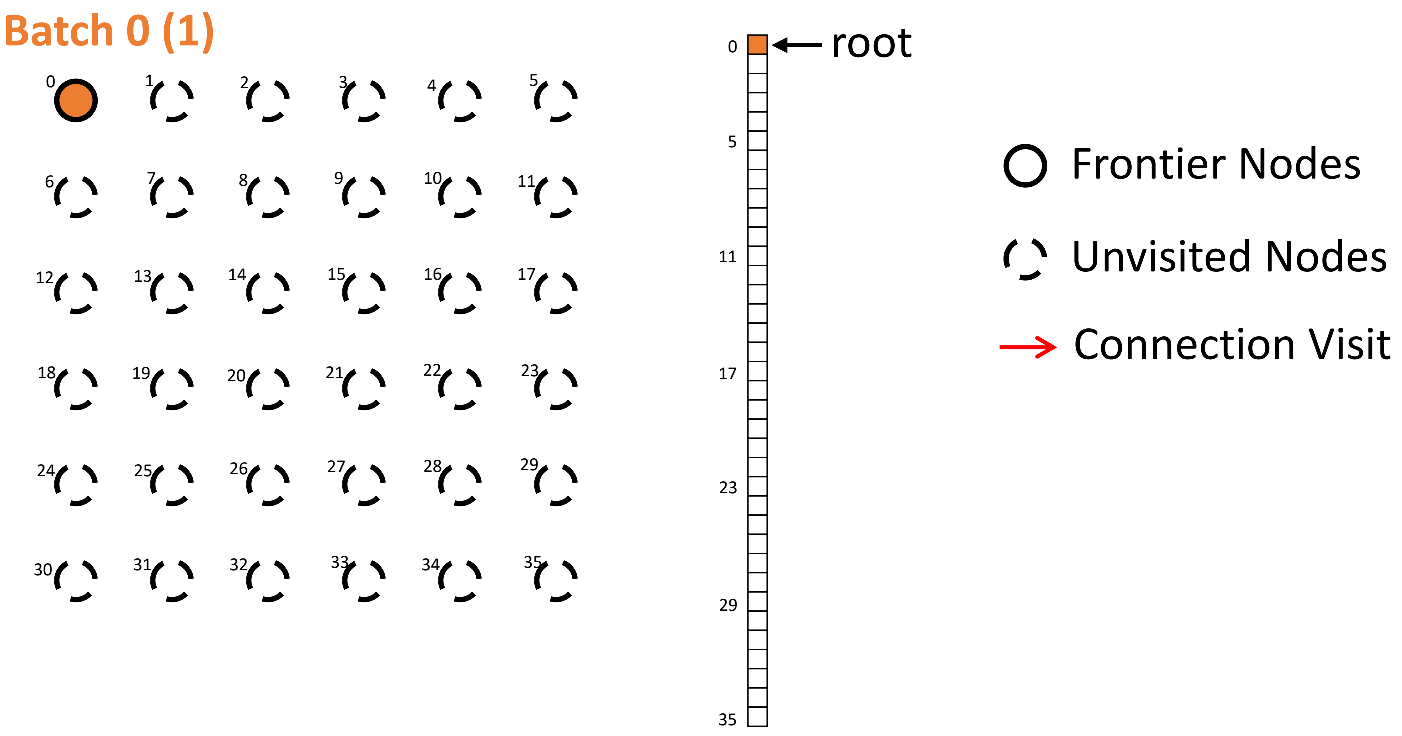
```
//SPMV
for(int m = 0; m < nump; m++){
  double reduce = 0;
  for(int n = displ[m]; n < displ[m+1]; n++)
    reduce += value[n]*x[index[n]];
  y[m] = reduce;
}
```

Easily Parallelized

**Other HPCG Operations**

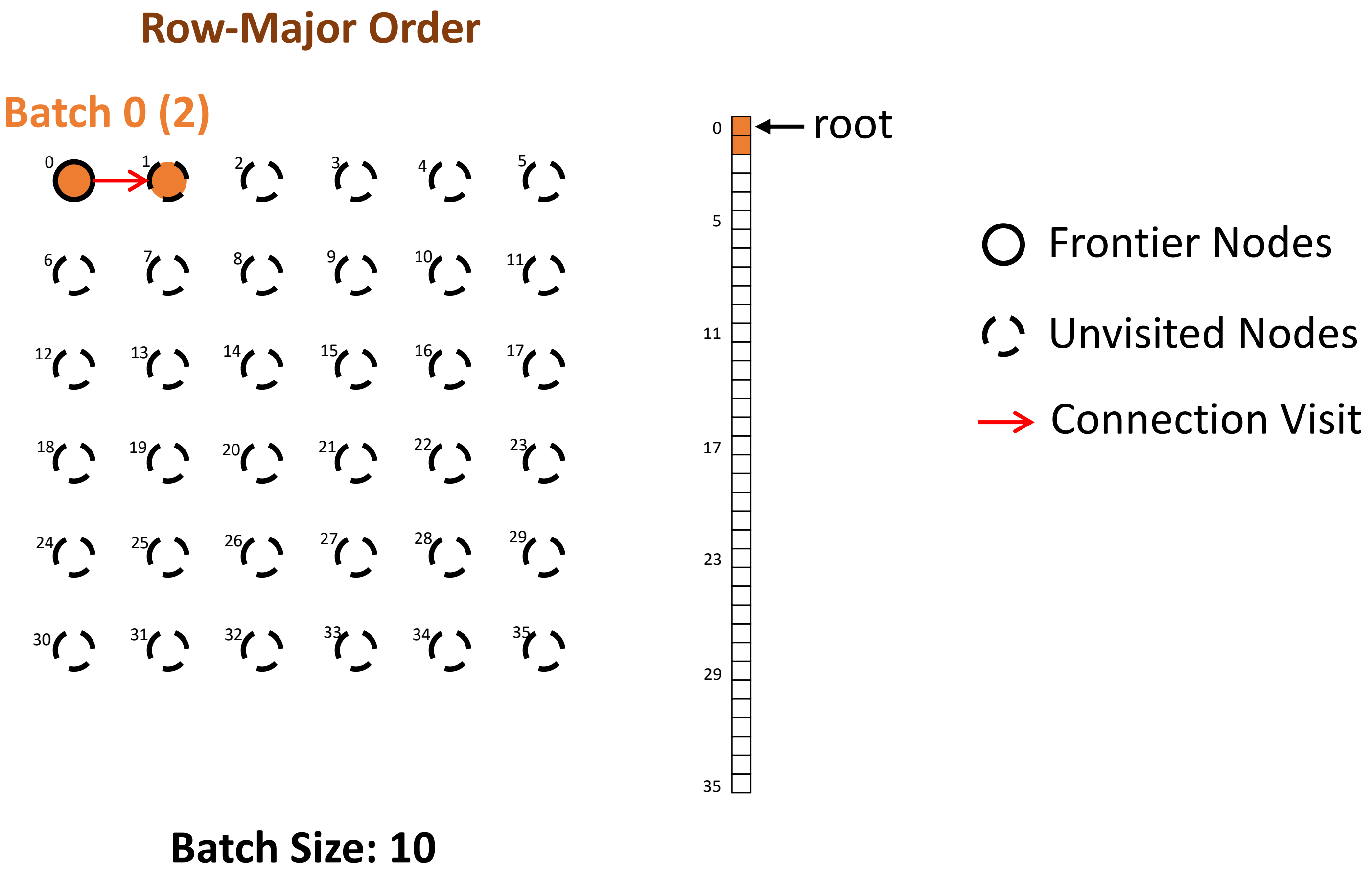
# Batching Algorithm

## Row-Major Order



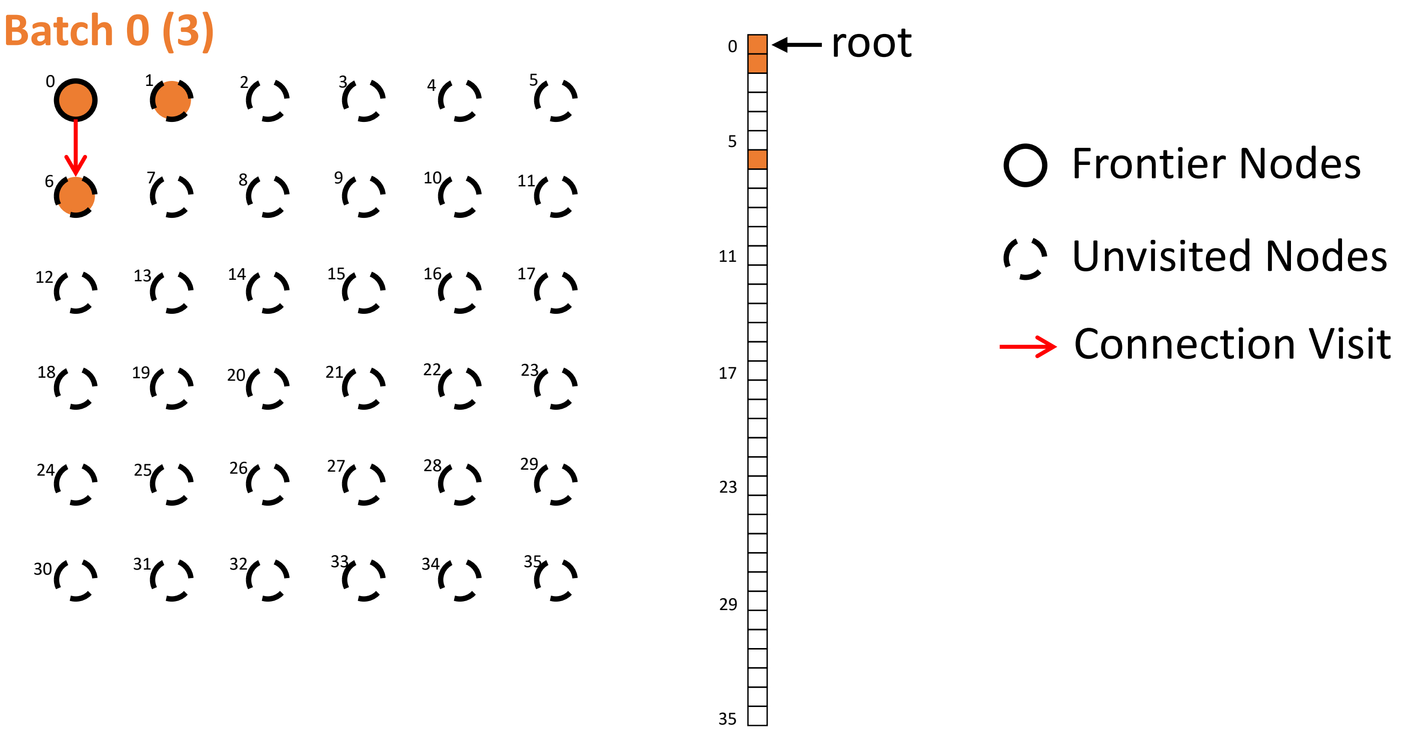

**Batch Size: 10**

# Batching Algorithm



# Batching Algorithm

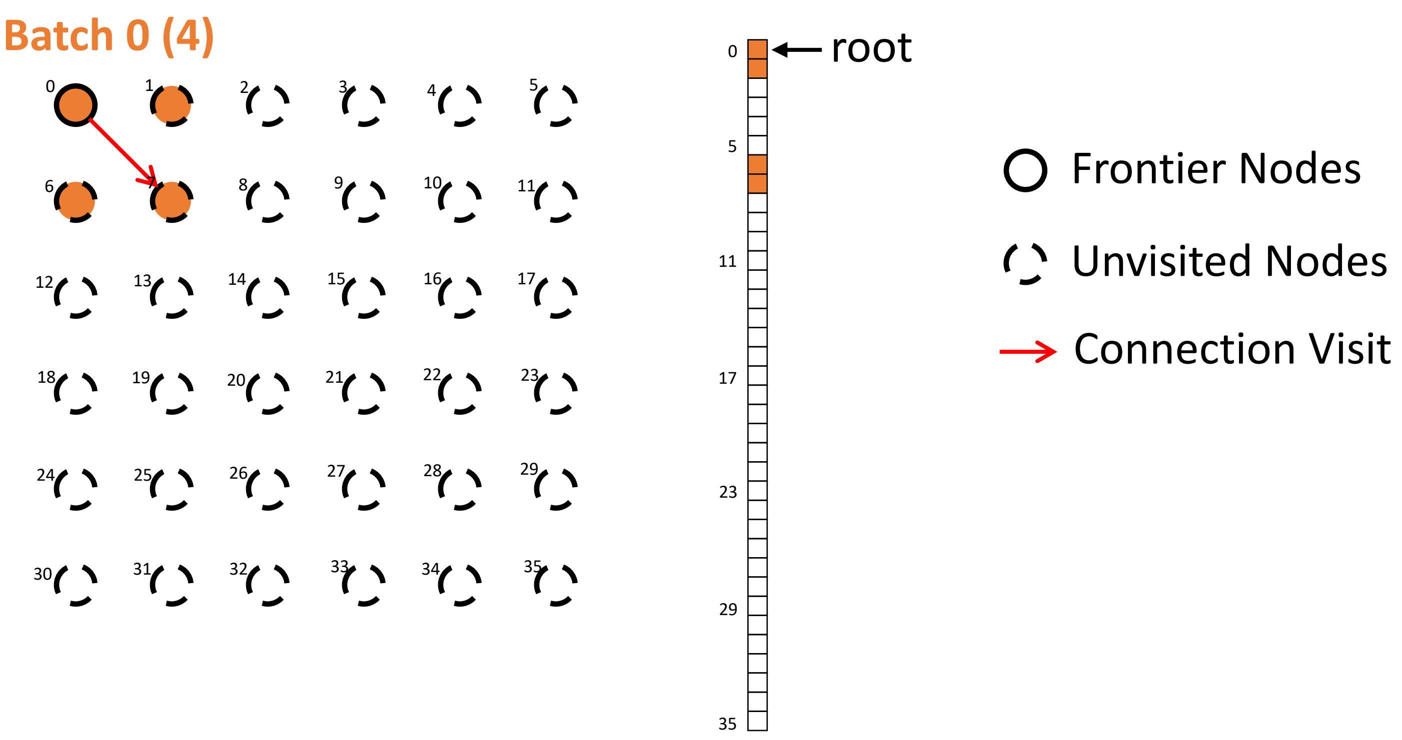
## Row-Major Order




**Batch Size: 10**

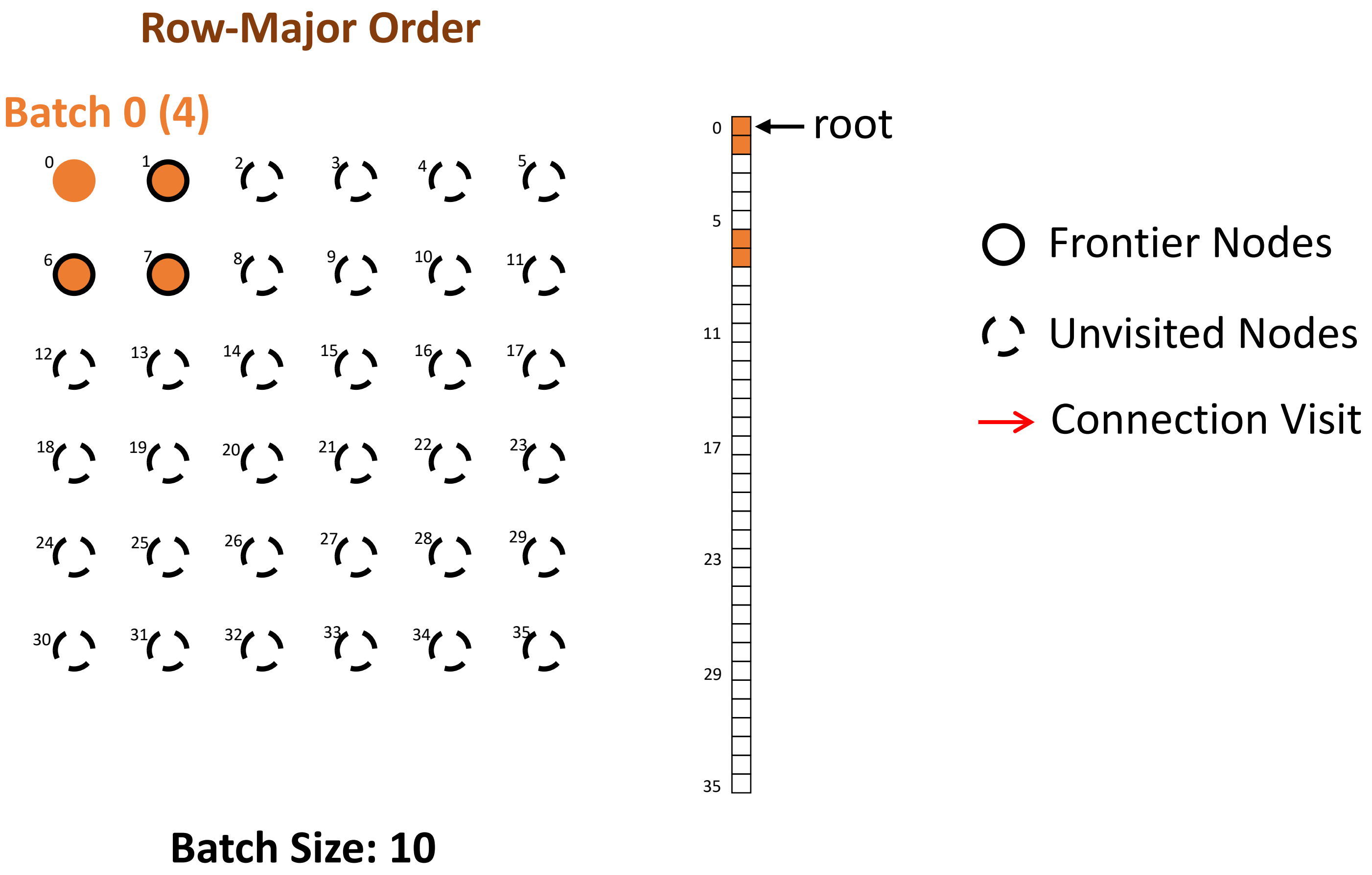
# Batching Algorithm

## Row-Major Order



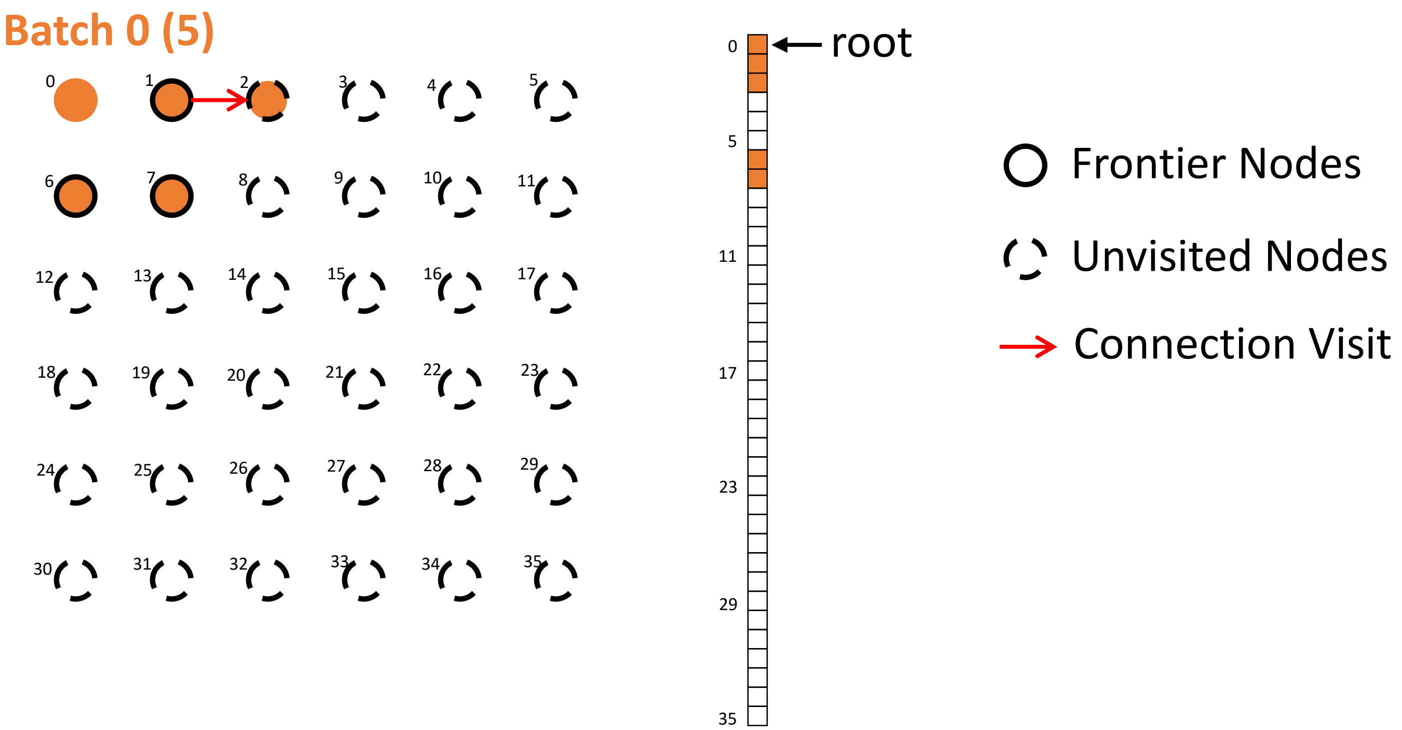

**Batch Size: 10**

# Batching Algorithm



# Batching Algorithm

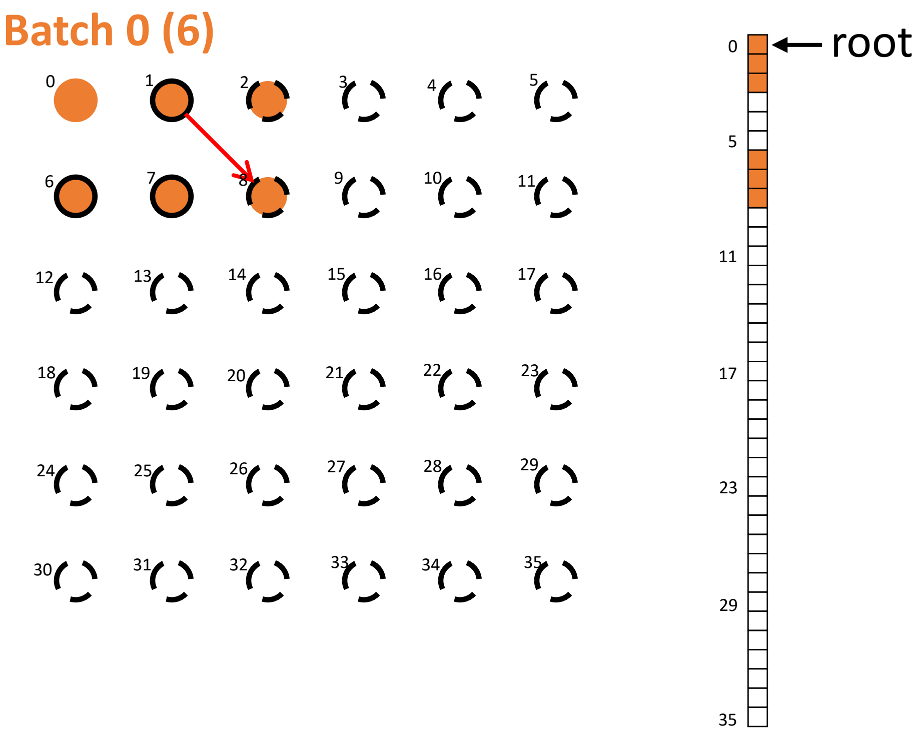
## Row-Major Order




**Batch Size: 10**

# Batching Algorithm

## Row-Major Order




Frontier Nodes

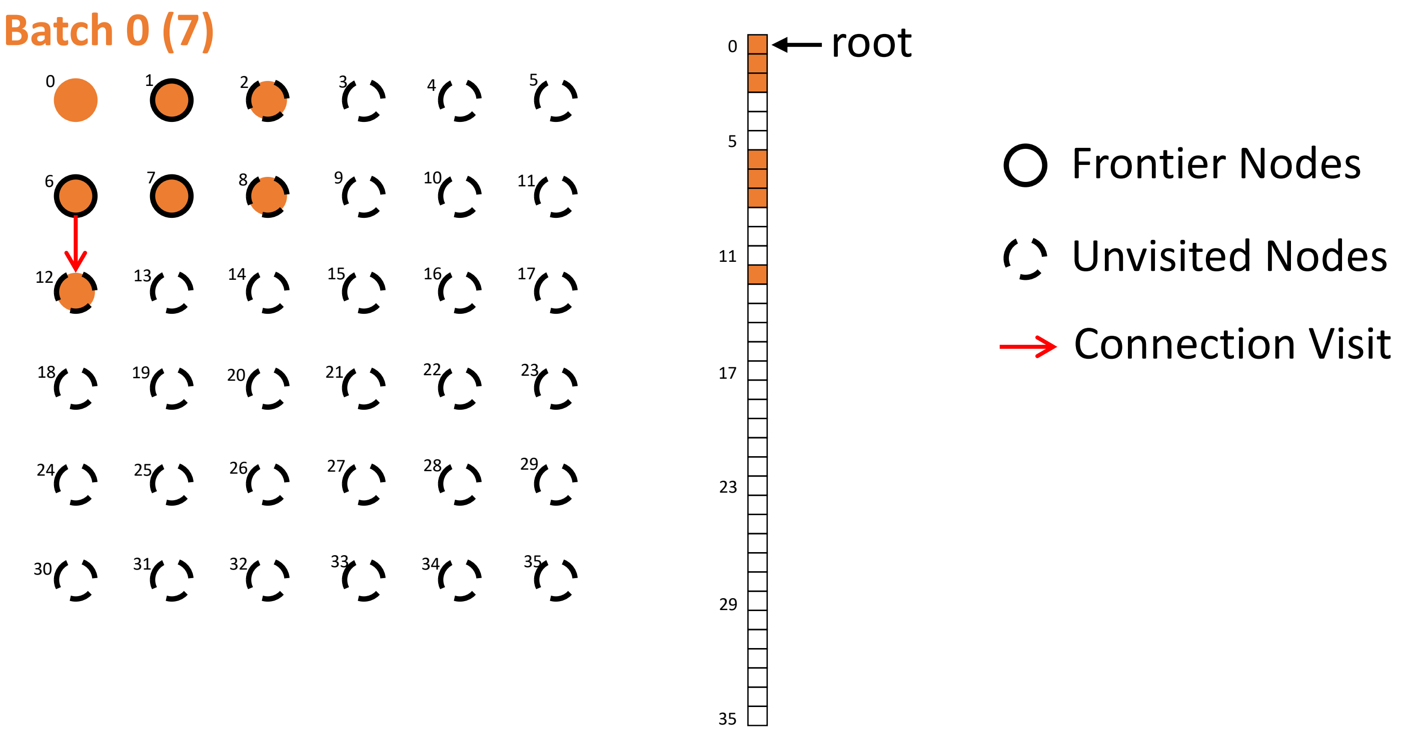
Unvisited Nodes

Connection Visit

**Batch Size: 10**

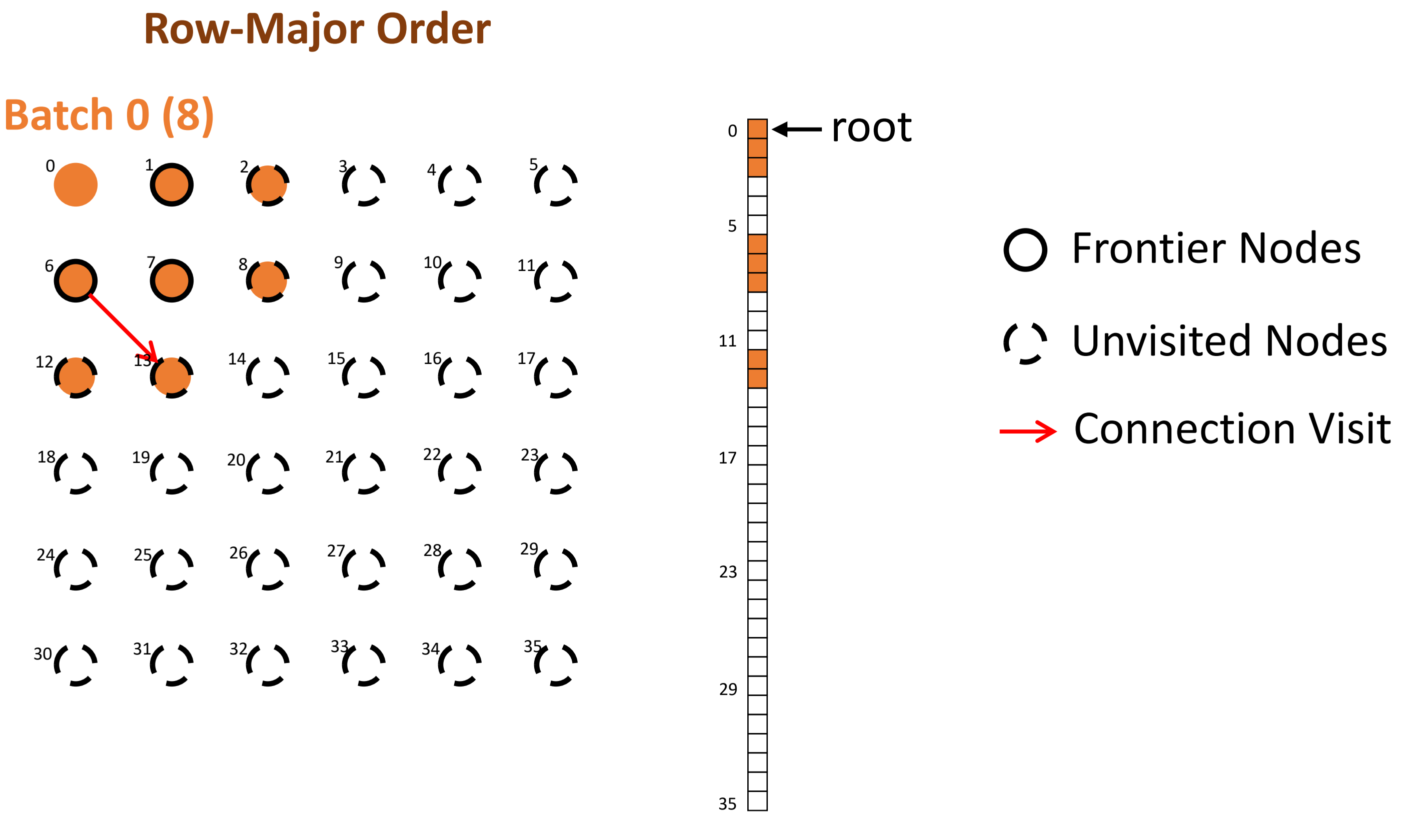
# Batching Algorithm

## Row-Major Order



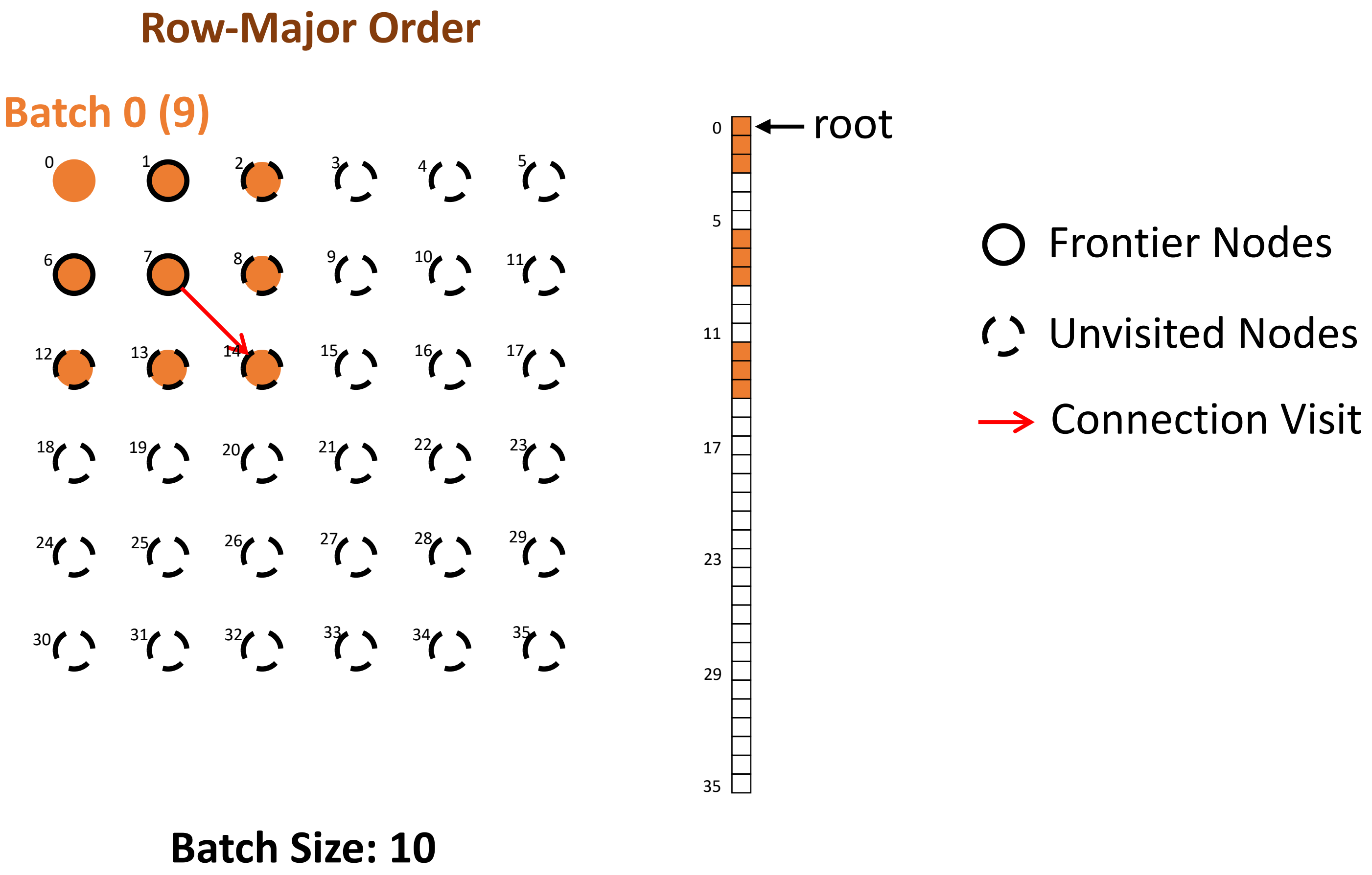

**Batch Size: 10**

# Batching Algorithm

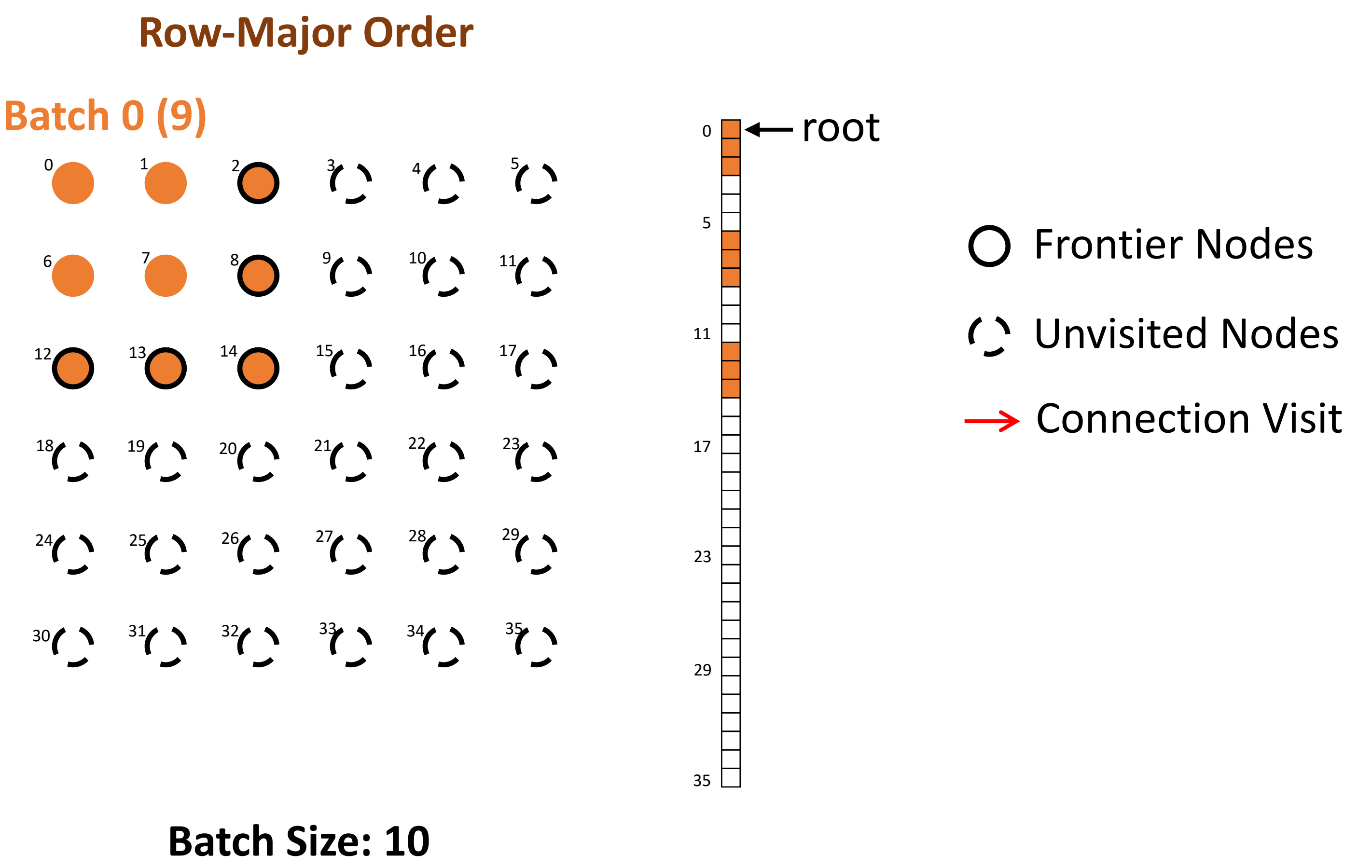



Batch Size: 10

# Batching Algorithm

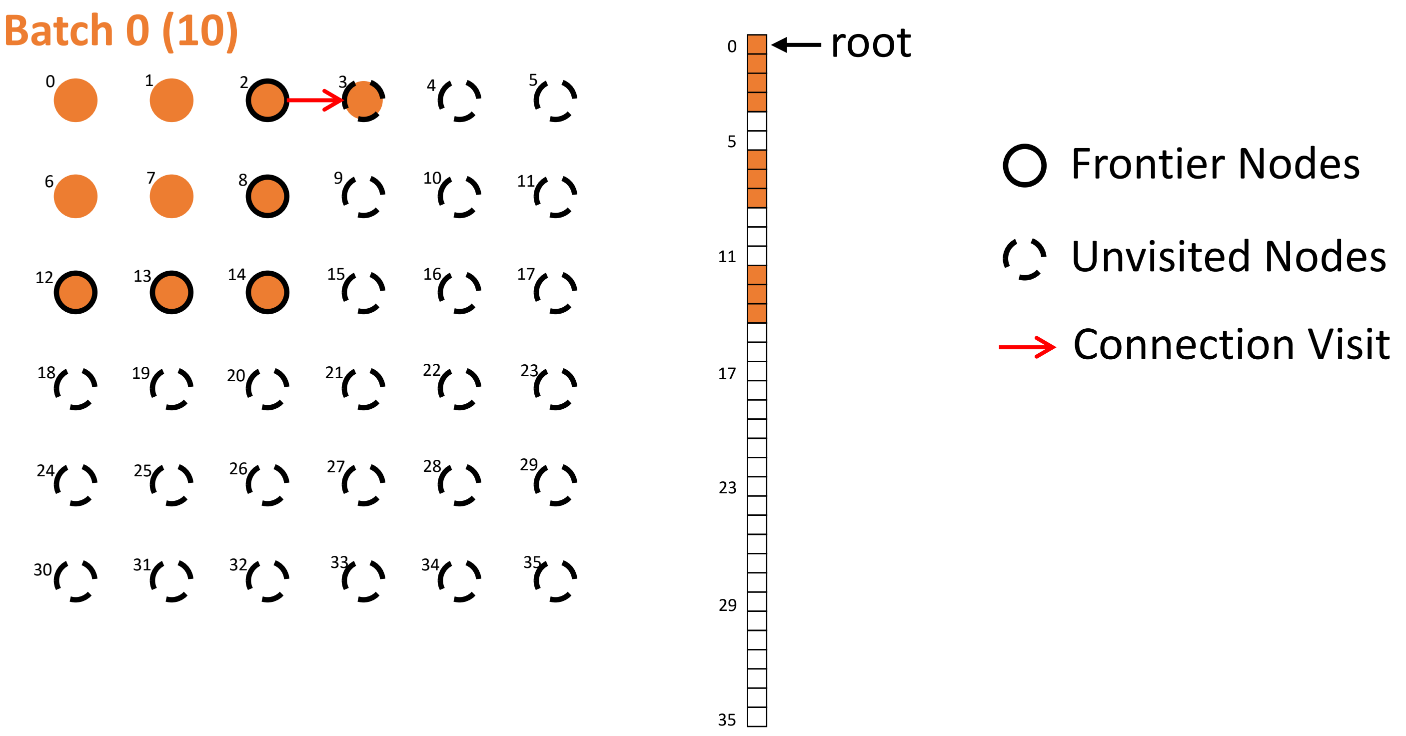


# Batching Algorithm



# Batching Algorithm

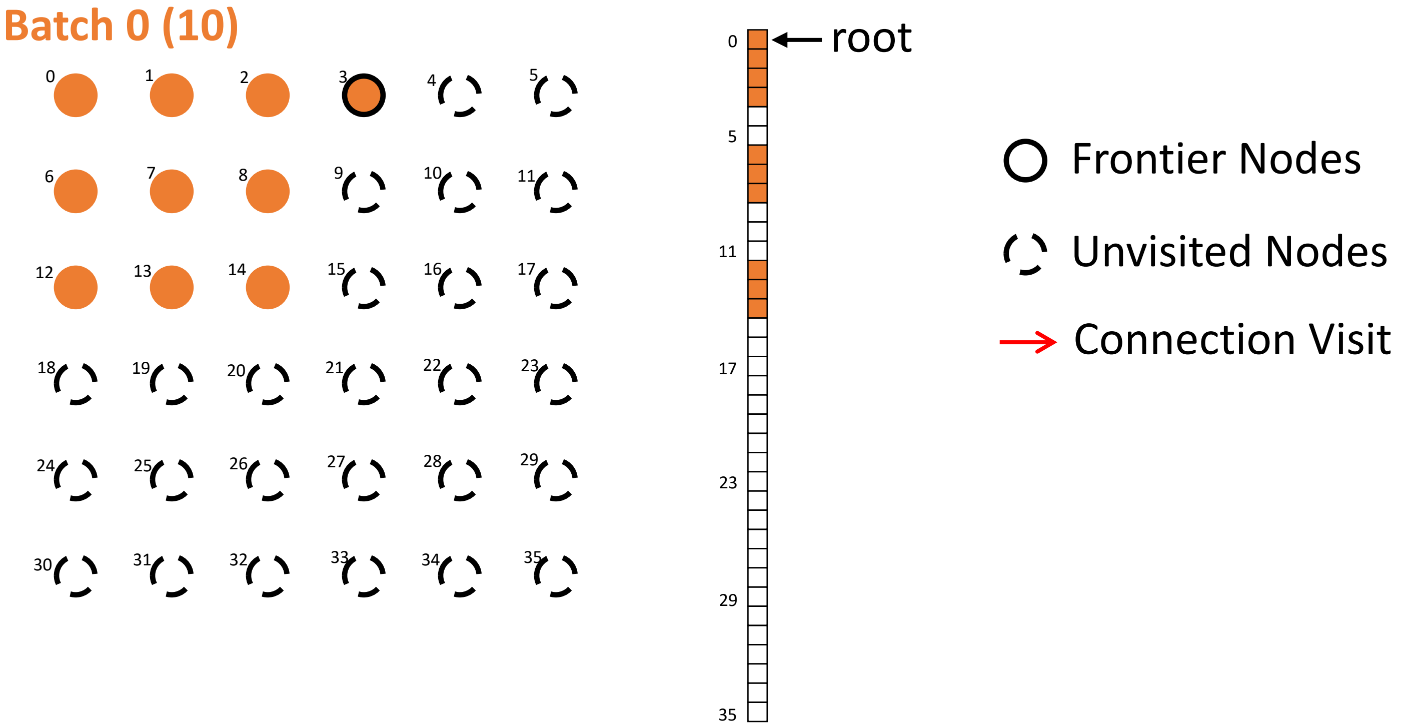
## Row-Major Order




**Batch Size: 10**

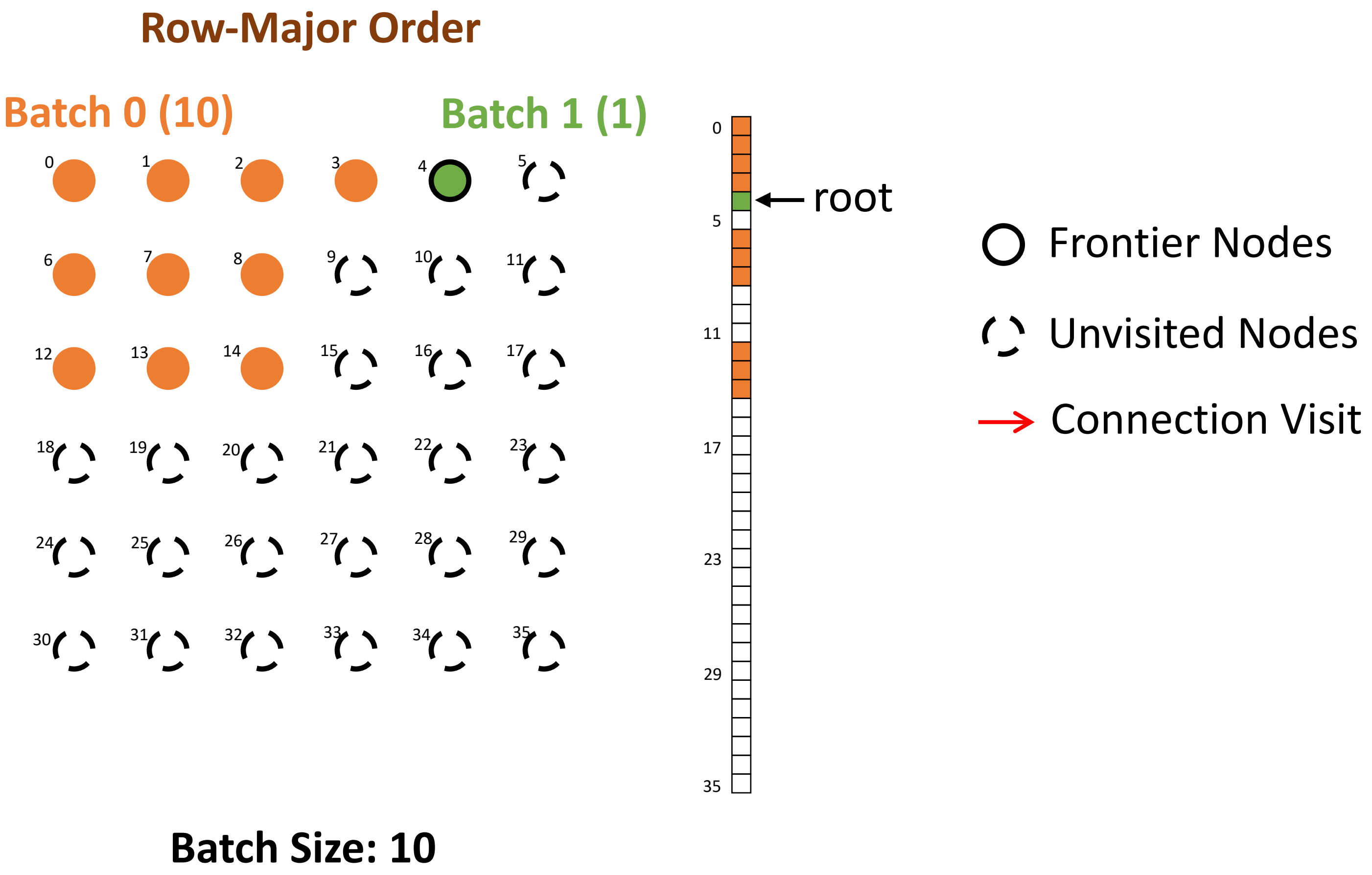
# Batching Algorithm

## Row-Major Order



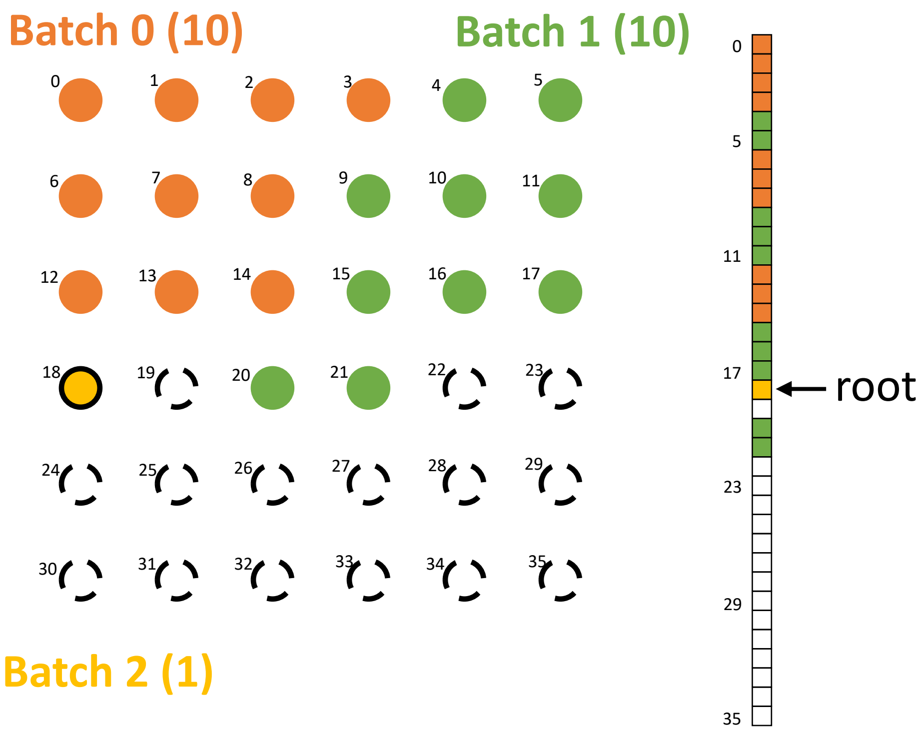

**Batch Size: 10**

# Batching Algorithm



# Batching Algorithm

## Row-Major Order



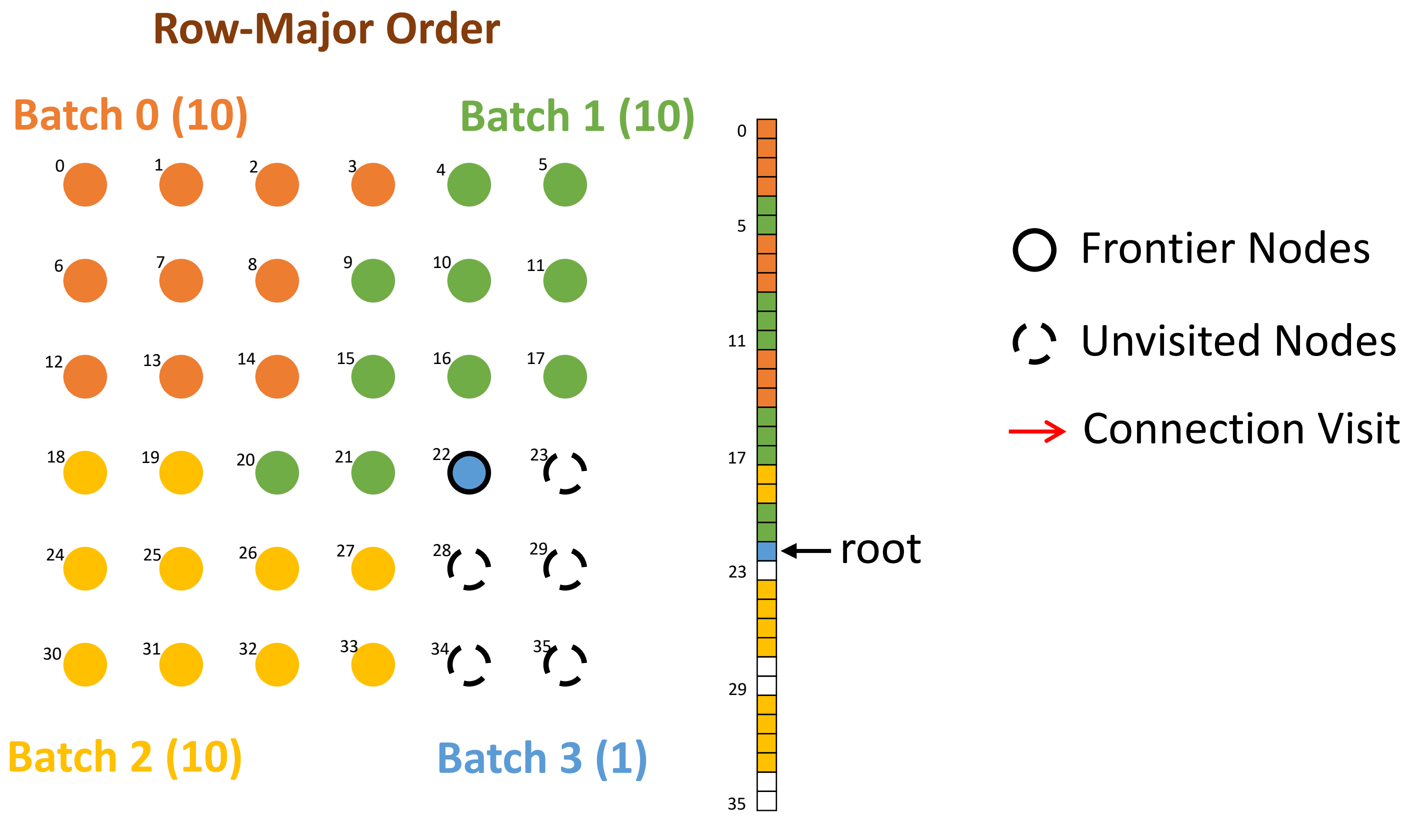

Frontier Nodes

Unvisited Nodes

Connection Visit

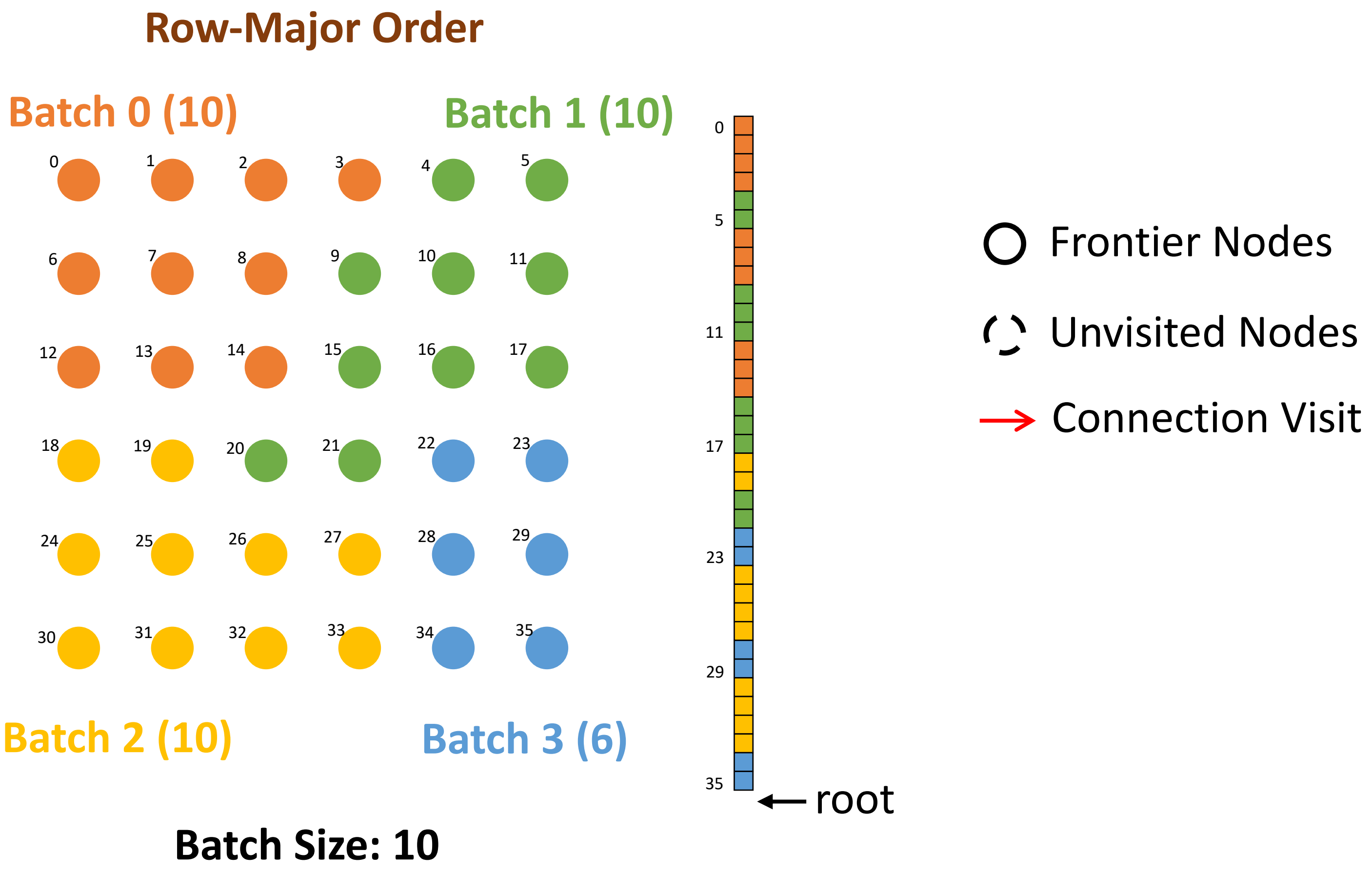
**Batch Size: 10**

# Batching Algorithm



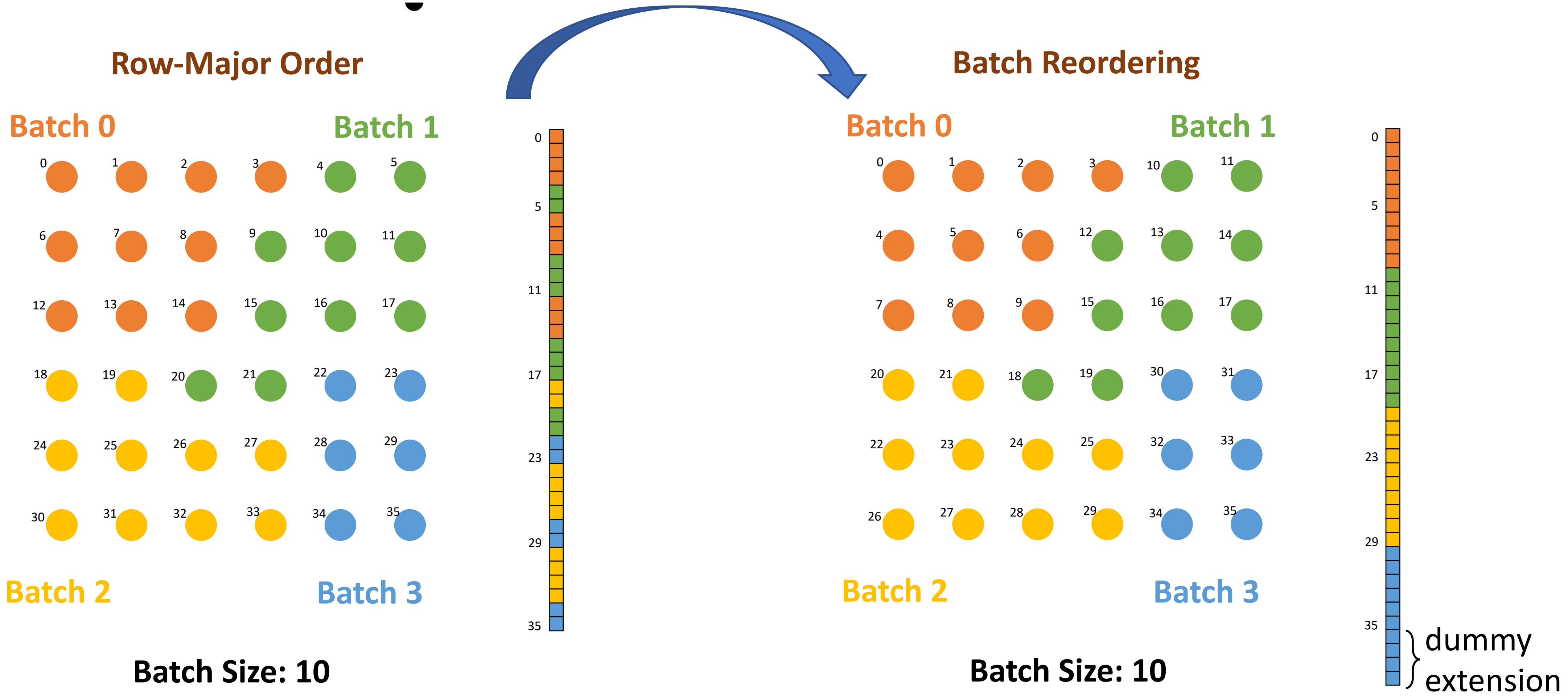

Batch Size: 10

# Batching Algorithm



**IDEA: Reorder to increase cache reuse**

Comput. & Memory Complexity: $\mathcal{O}(N)$



## Original HPCG Gauss-Seidel

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int m = 0; m < nump; m++){
  double reduce = b[m];
  for(int n = displ[m]; n < displ[m+1]; n++)
    reduce -= value[n]*x[index[n]];
  x[m] += reduce/diag[m];
}
```

## Memory Transactions

**Auxiliary Data:** $3N \times 8$ Bytes Read

**Matrix Data:** $27N \times (8+4)$ Bytes Read

**Vector Data:** $27N \times 8$ Bytes Read ***Irregular Access***

$N \times 8$ Bytes Read

$N \times 8$ Bytes Write

## Gauss-Seidel with Reordering

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int m = 0; m < nump; m++){
  double reduce = b[m];
  for(int n = displ[m]; n < displ[m+1]; n++)
    reduce -= value[n]*x[index[n]];
  x[m] += reduce/diag[m];
}
```

**No kernel code change!**

## Memory Transactions

**Auxiliary Data:** $3N \times 8$ Bytes Read

**Matrix Data:** $27N \times (8+4)$ Bytes Read

**Vector Data:** $\boxed{27N \times 8 \text{ Bytes Read}}$ **Maximum cache-reuse: 27**

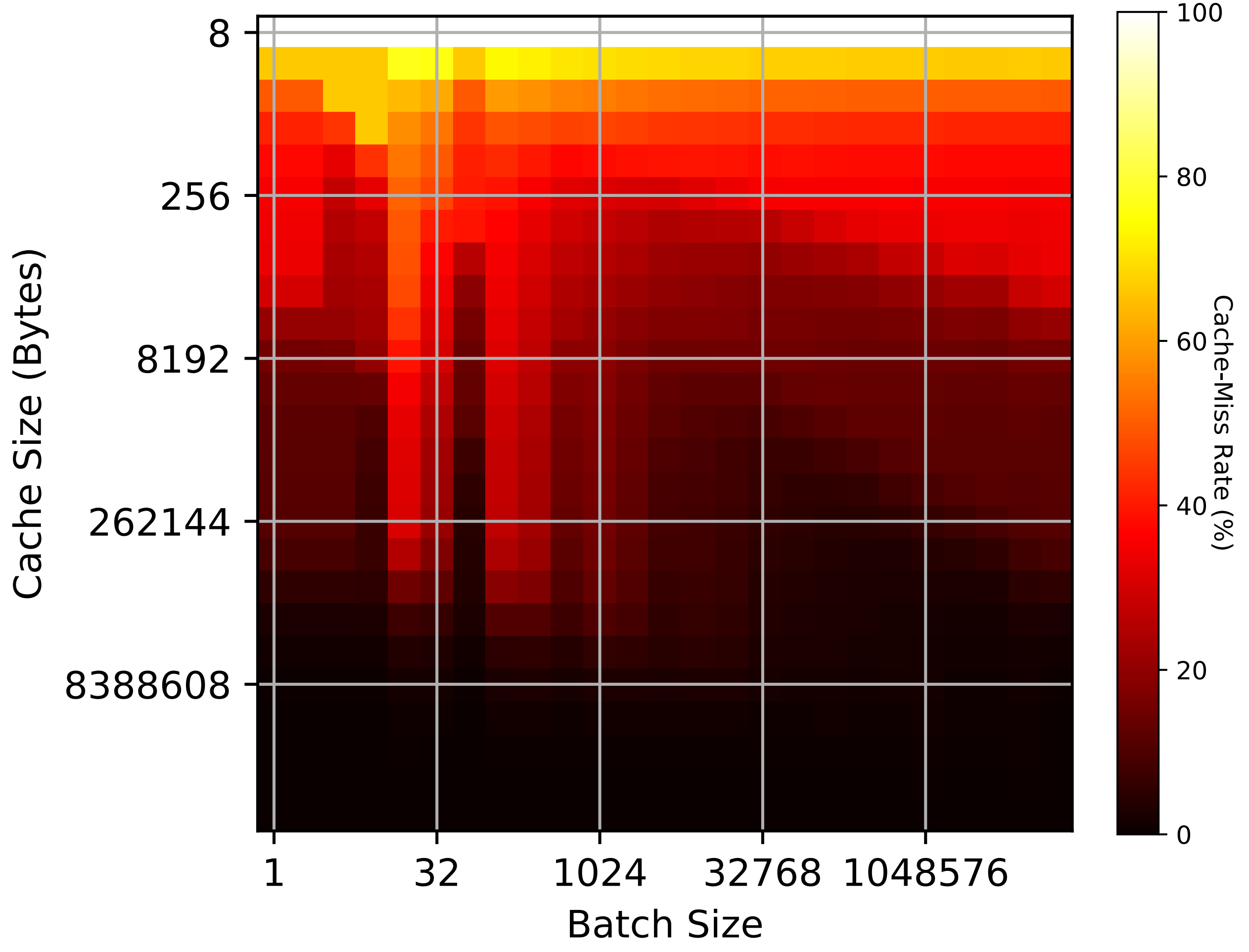
$N \times 8$ Bytes Read

$N \times 8$ Bytes Write

**IDEA: Use a buffer for explicit vector data reuse**

**IDEA: Use short index to save memory B/W**

## Cache-Miss Rate Simulation (for single-level and single-line cache)

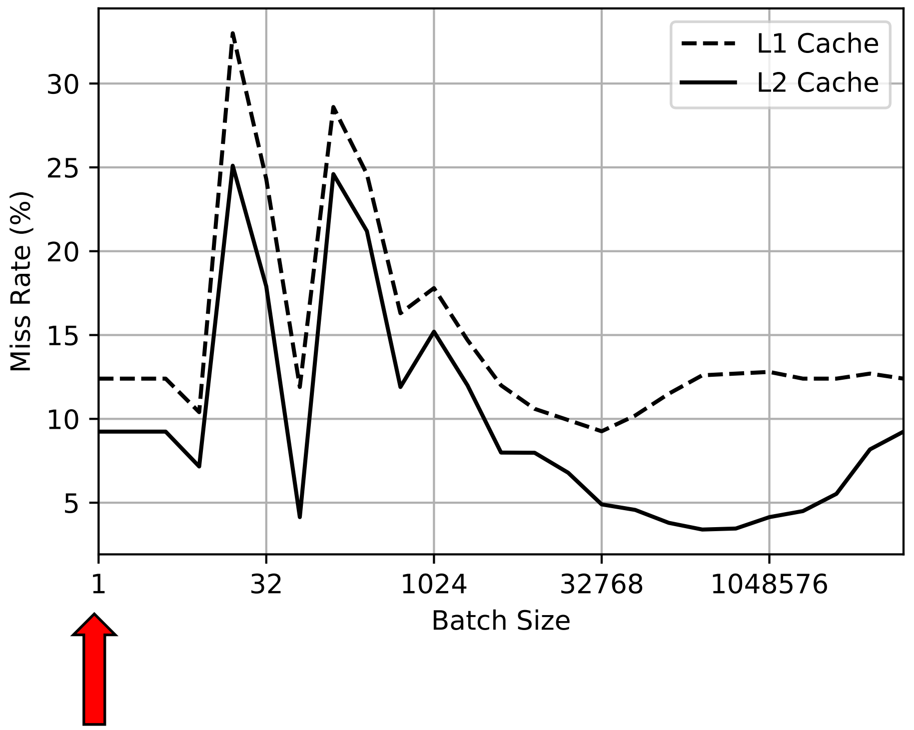



**POWER9 Specs**

L1: 32 KB per Core

L2: 512 KB per Core

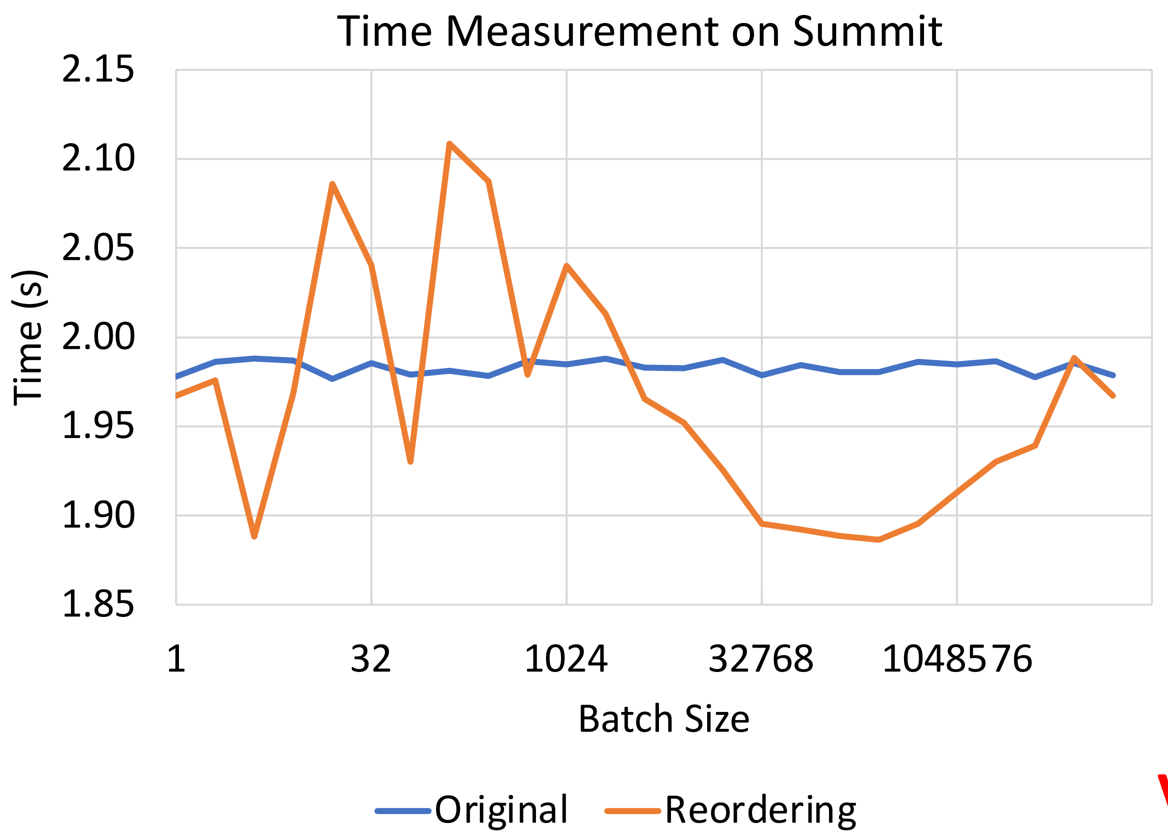
L3: 120 MB per Chip

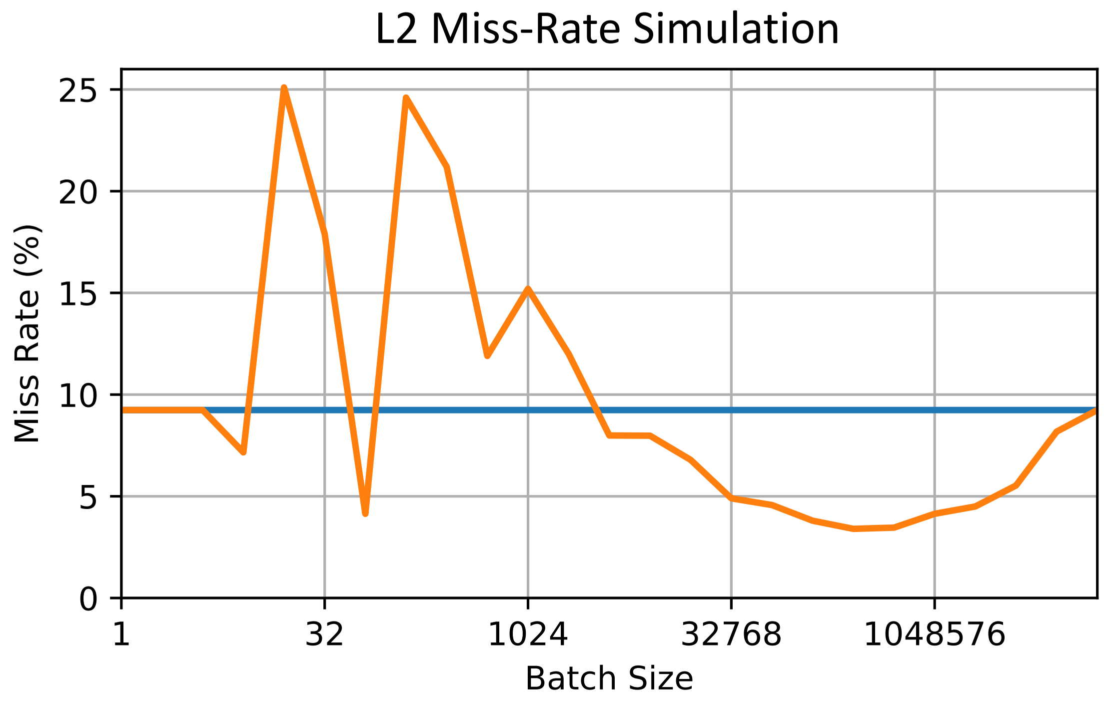



Batch Size = 1 means no reordering

# Cache-Miss Rate Simulation vs. Time Measurements

220x220x220 Grid with 27-Point Stencil



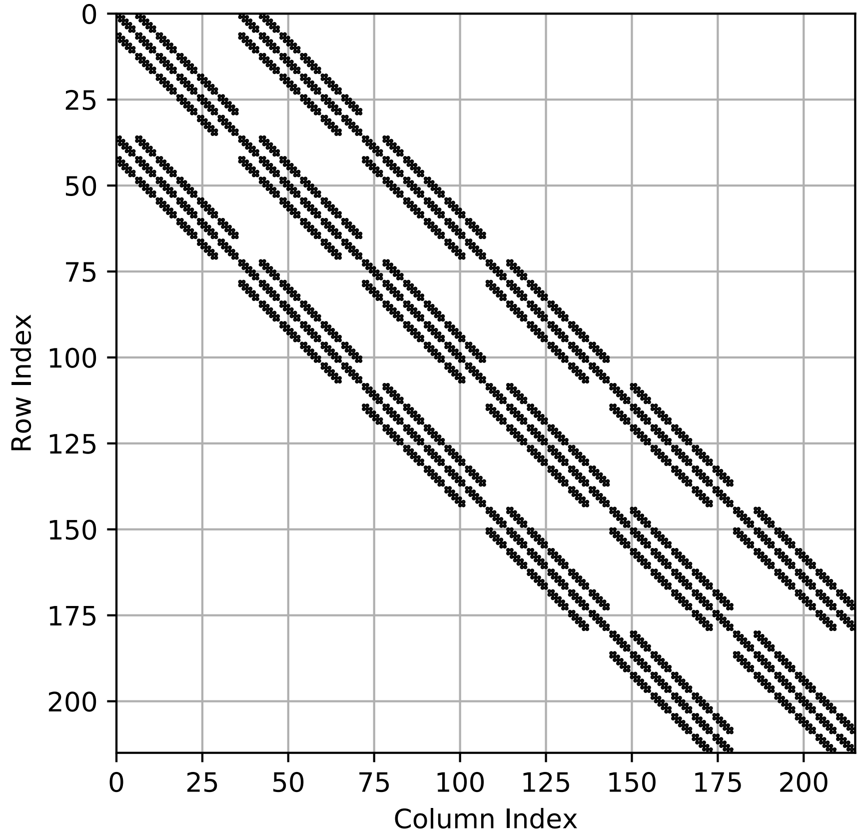




**Very Good Match!!!**

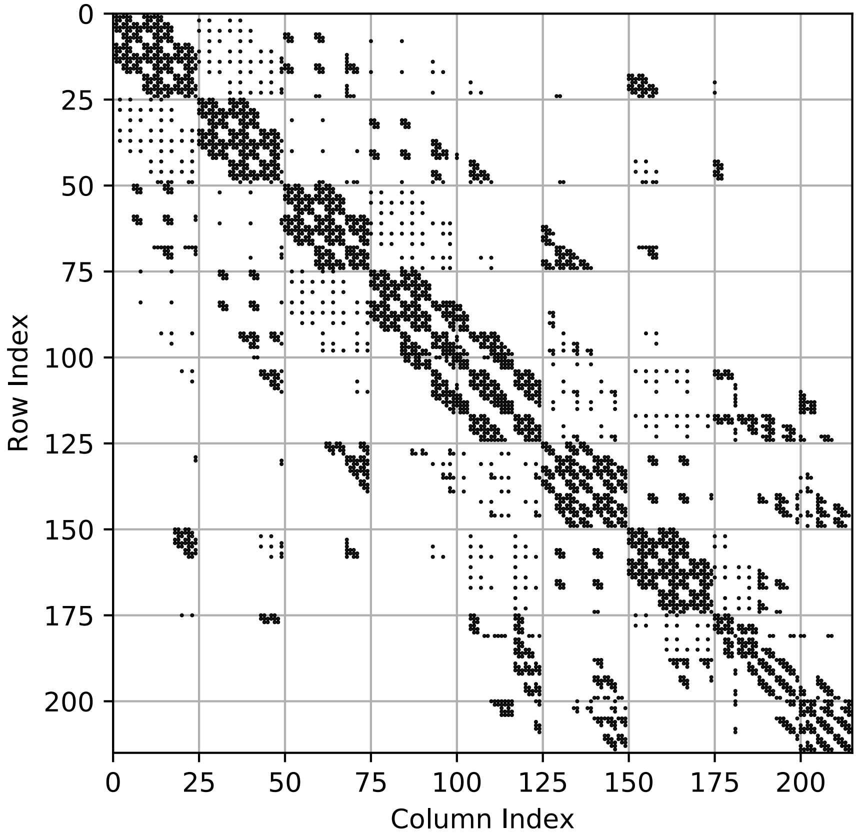
## Matrix Sparsity Pattern

6x6x6 27-point stencil
Batch Size: 25

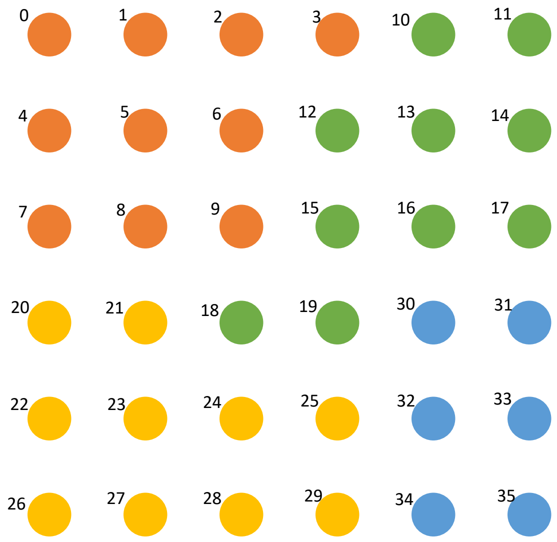
### Original

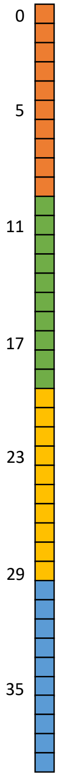



### Batch Reordering



# Buffering Algorithm

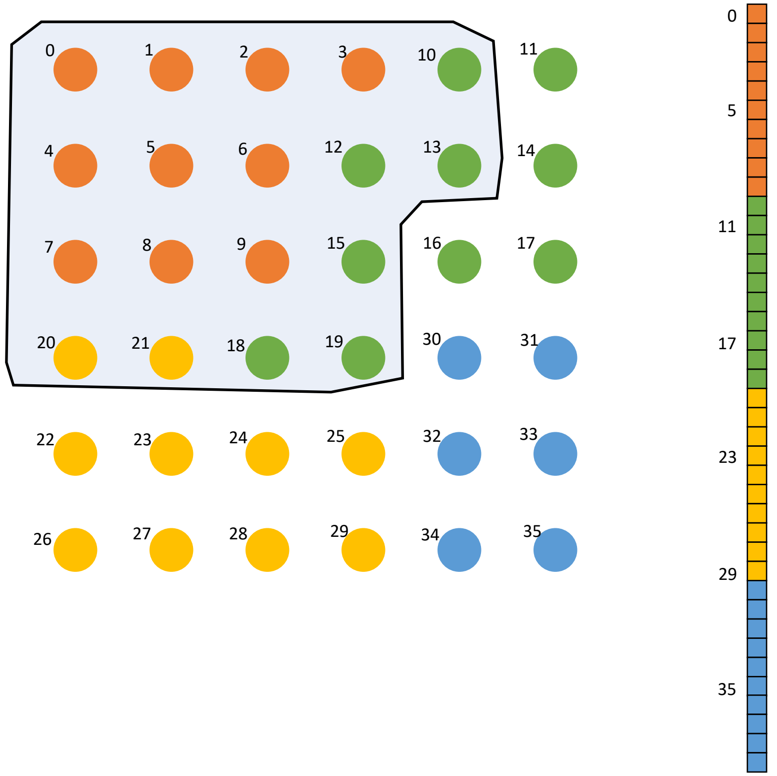



```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){

}
```

# Buffering Algorithm

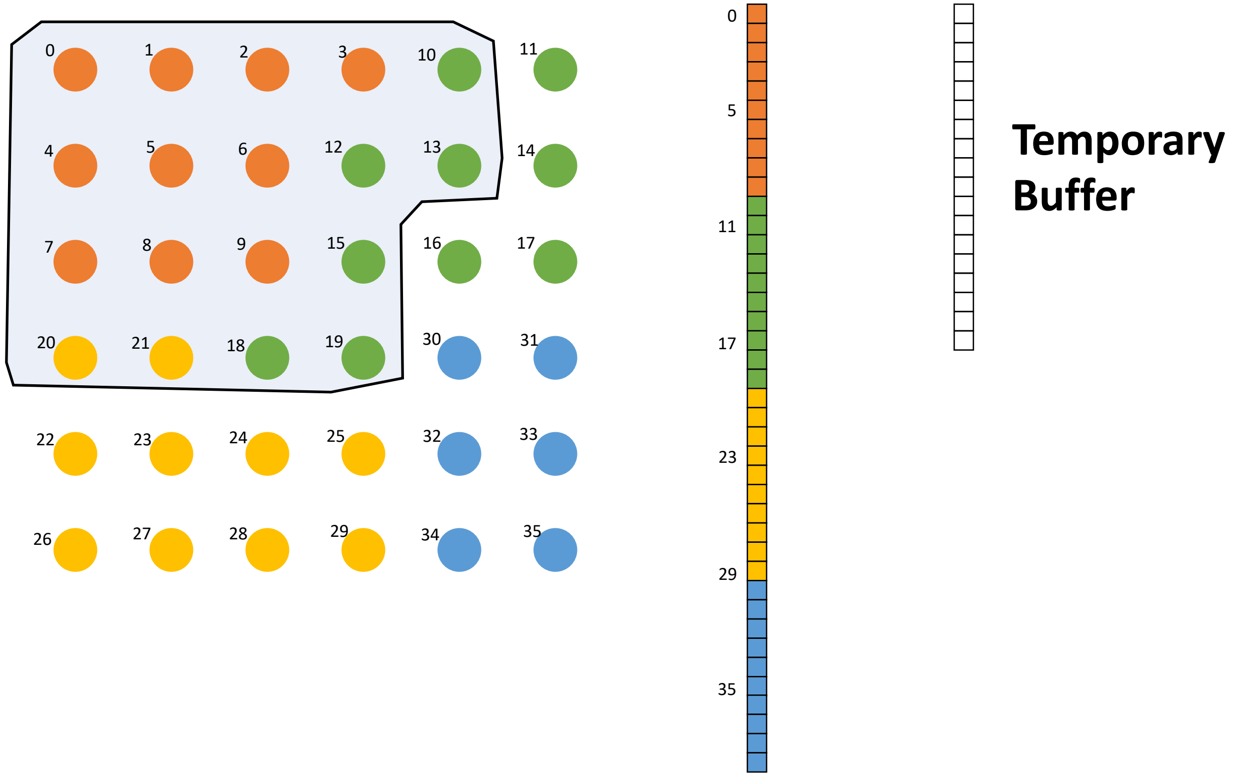
**Data Accessed by Batch 0**




```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;



}
```

# Buffering Algorithm

Data Accessed by Batch 0



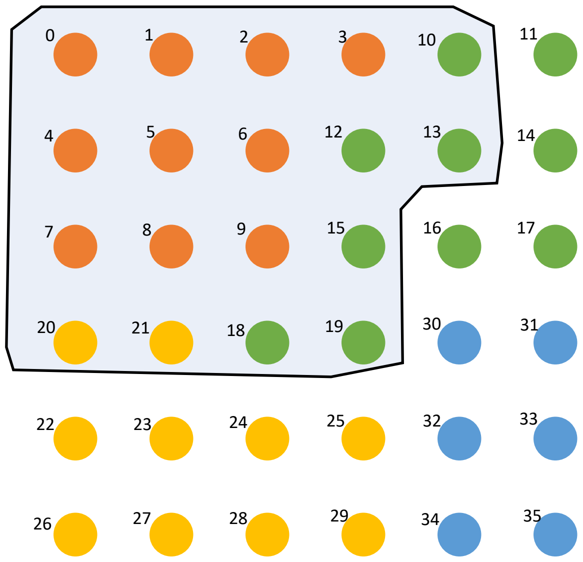

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];



















}
```

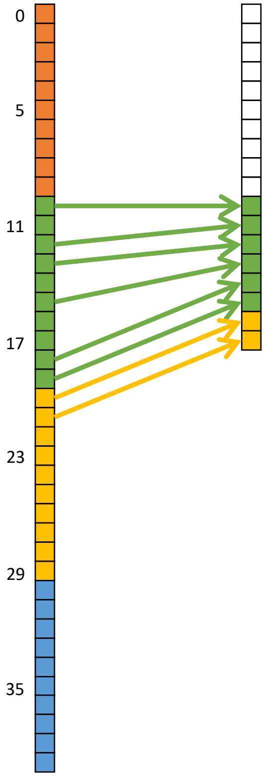
# Buffering Algorithm

Load inter-batch data

- Irregular Access
- Hardly Vectorizable
- Requires a mapping (additional overhead)

**Data Accessed by Batch 0**




**Temporary Buffer**

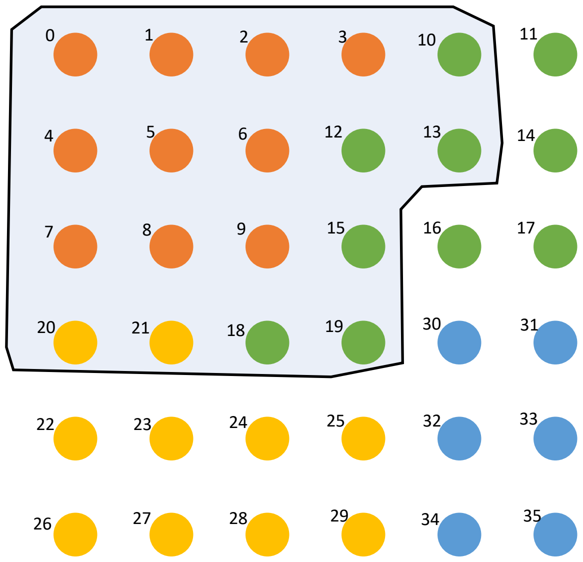
```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];









}
```

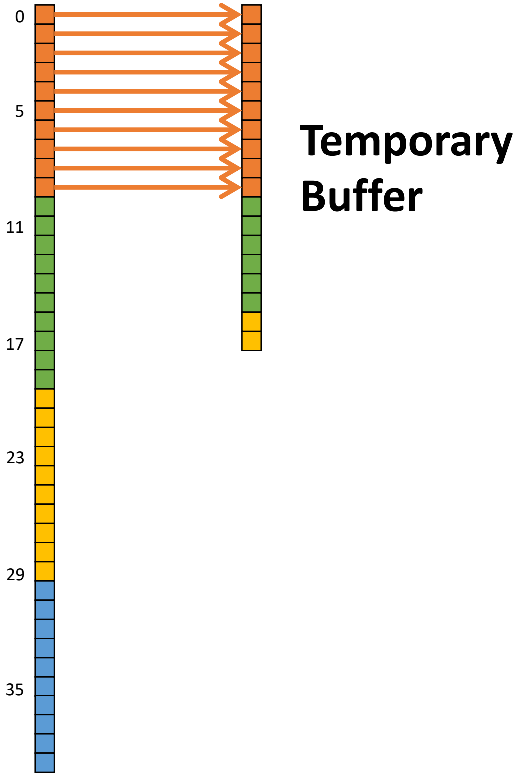
# Buffering Algorithm

Load intra-batch data

- Regular Access
- Vectorizable

**Data Accessed by Batch 0**



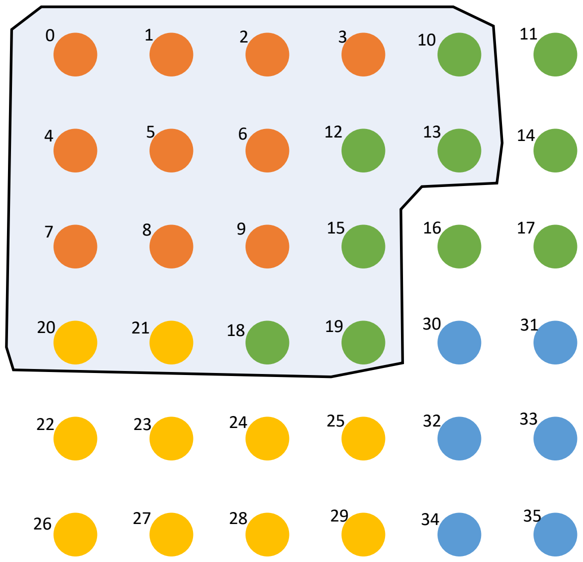

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
}
```

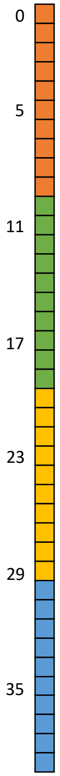
# Buffering Algorithm

Perform Gauss-Seidel from buffer

- Intra-Batch Data is Updated

**Data Accessed by Batch 0**




**Temporary Buffer**

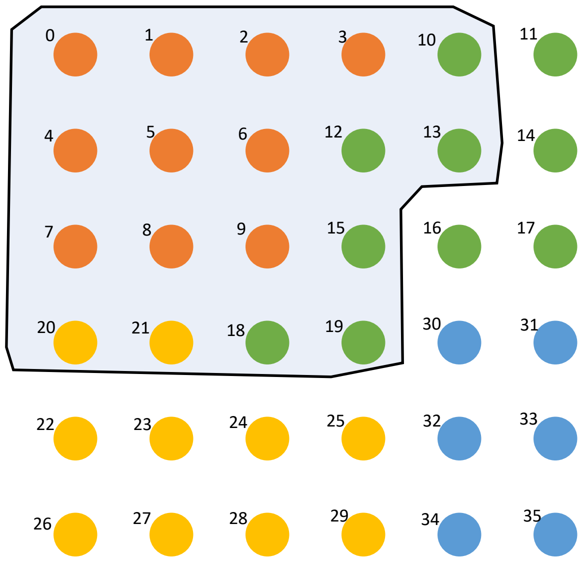
```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[index[n]];
    buffer[m-start] += reduce/diag[m];
  }


}
```

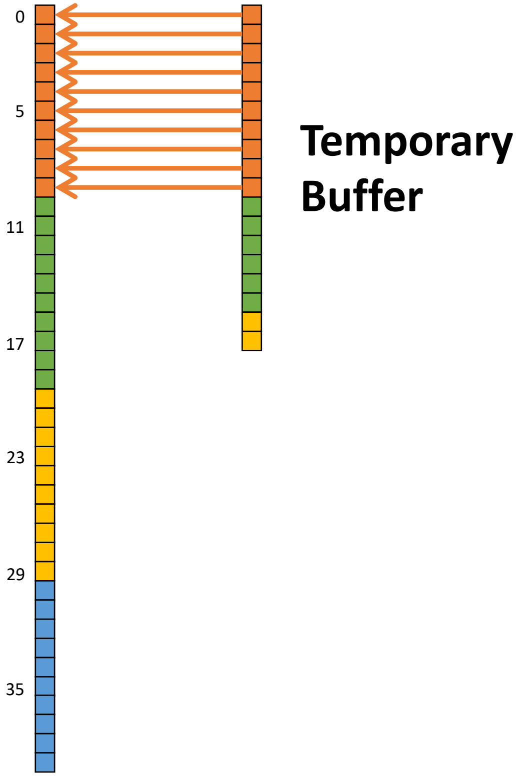
# Buffering Algorithm

Write-back intra-batch data

- Regular Access
- Vectorizable

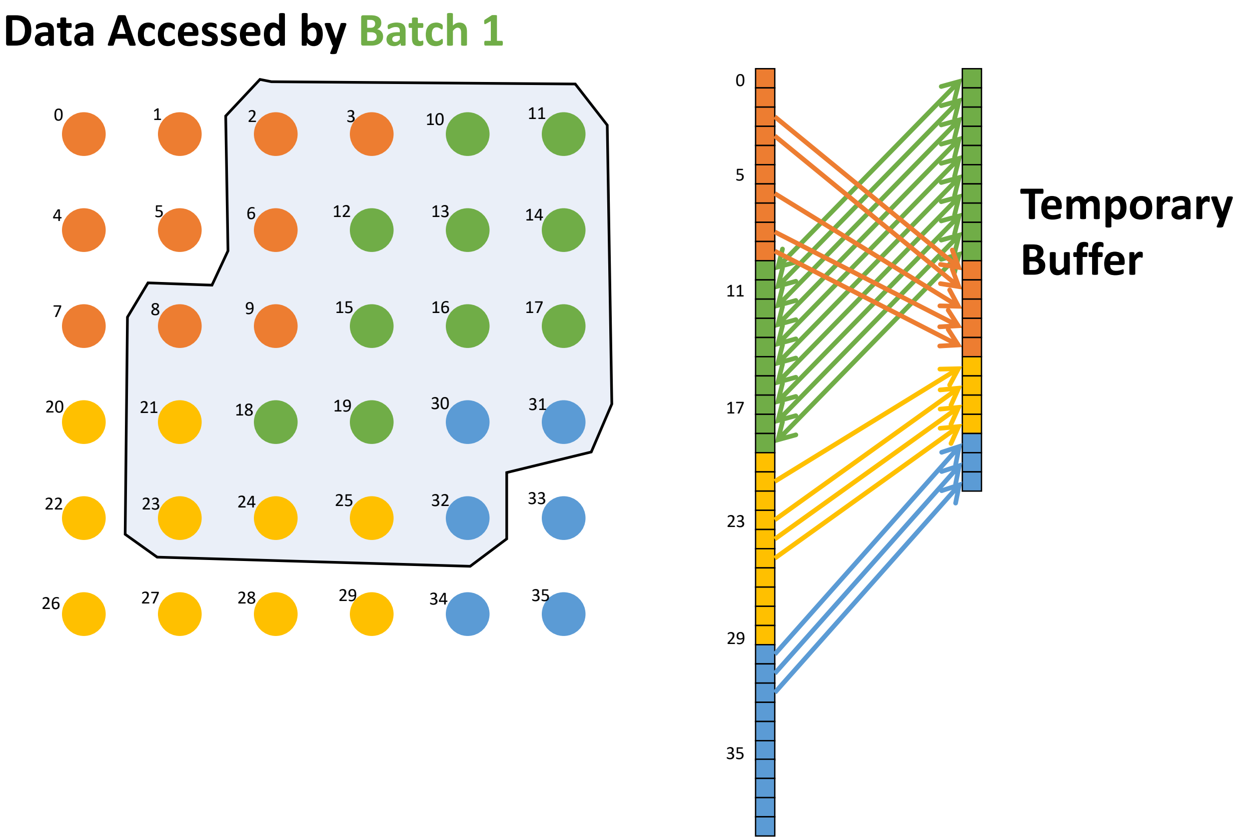
**Data Accessed by Batch 0**



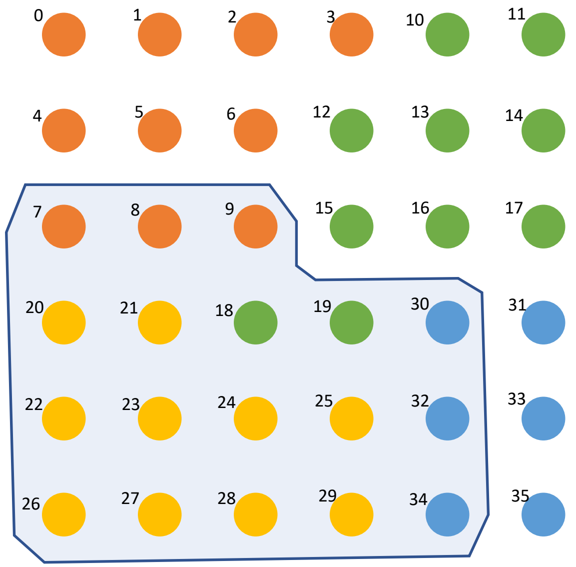

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[index[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

# Buffering Algorithm

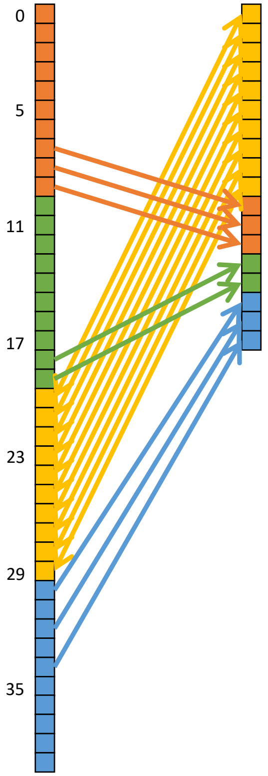



```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[index[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

# Buffering Algorithm



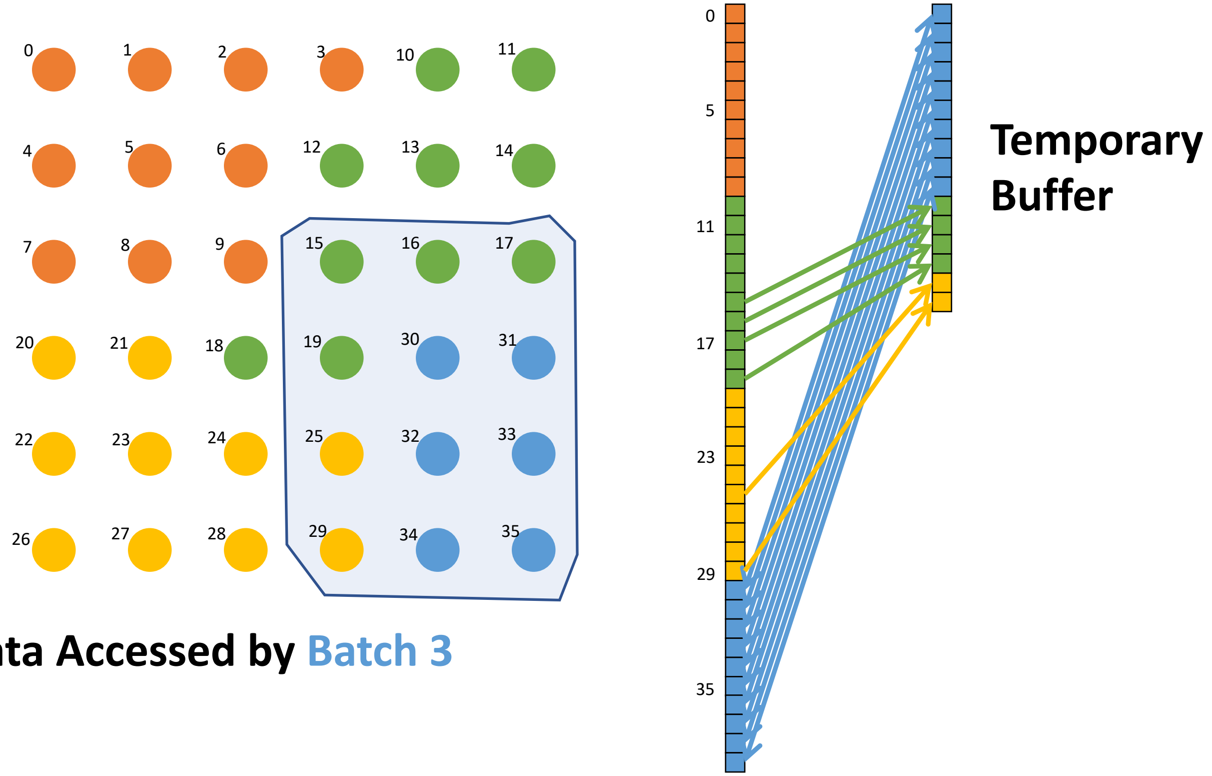

Data Accessed by Batch 2




**Temporary Buffer**

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[index[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

# Buffering Algorithm



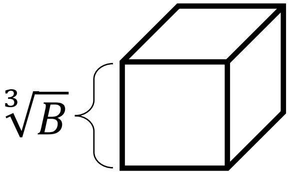

**Data Accessed by Batch 3**

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[index[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[index[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

**Overhead Analysis**

$B$: Batch Size

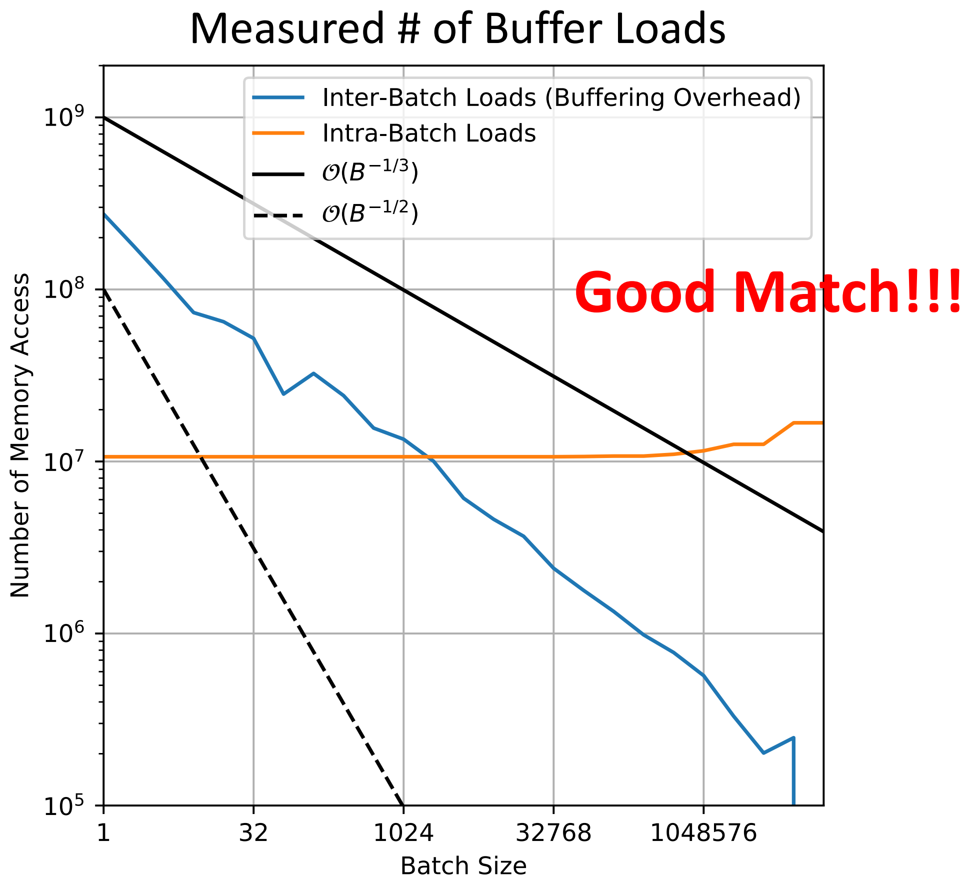



**Inter-Batch Loads per Batch:** $\mathcal{O}(B^{2/3})$

**Number of Batches:** $N/B$

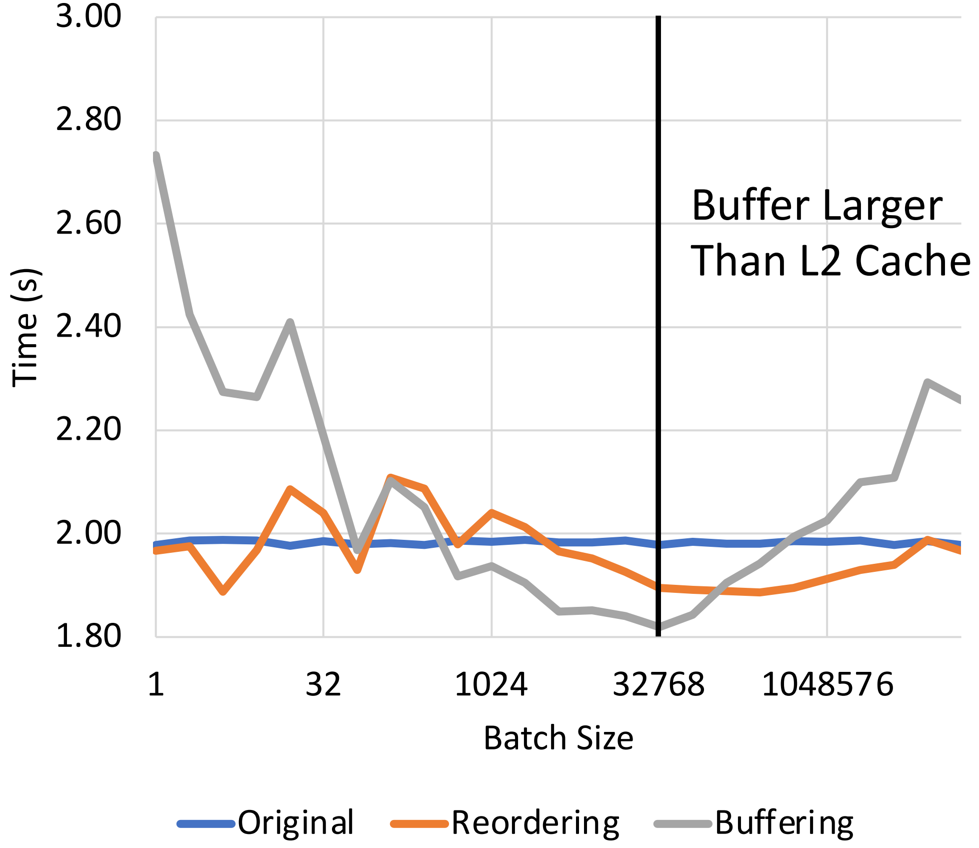
**Total Inter-Batch Loads:** $\mathcal{O}(NB^{-1/3})$

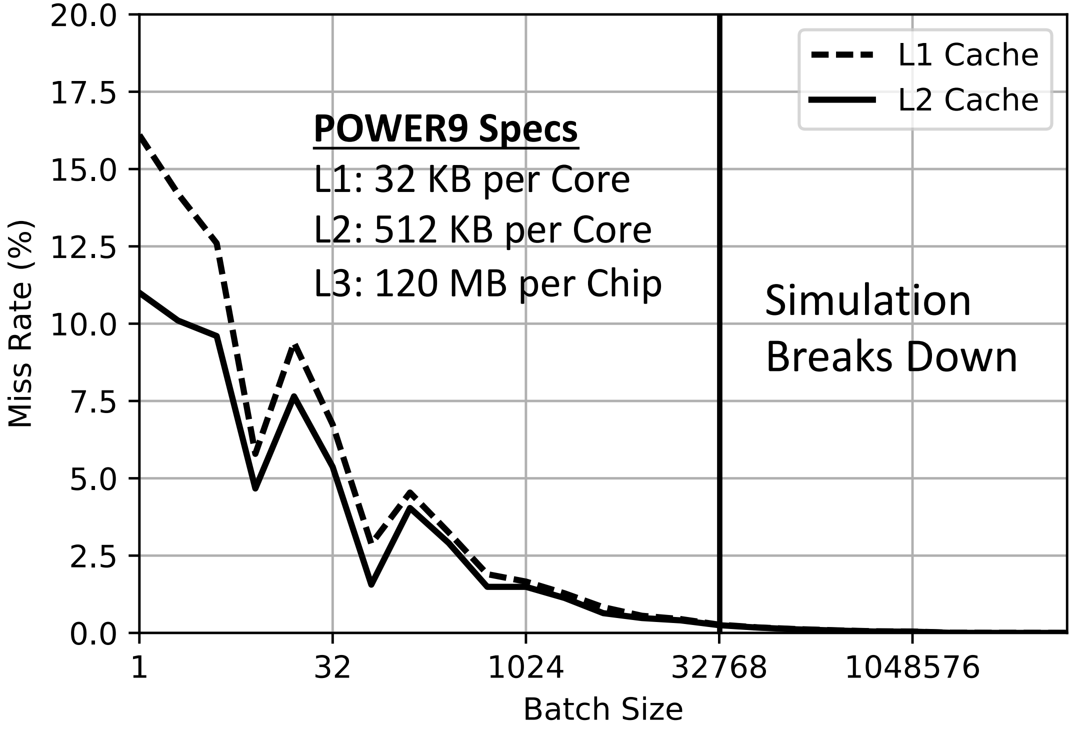
These are Irregular Access!



# Cache-Miss Rate Simulation
# (for single-level and single-line cache)

220x220x220 Grid with 27-Point Stencil



# Using Small Buffer

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

Use `unsigned short` instead of `int`

# Memory Transactions

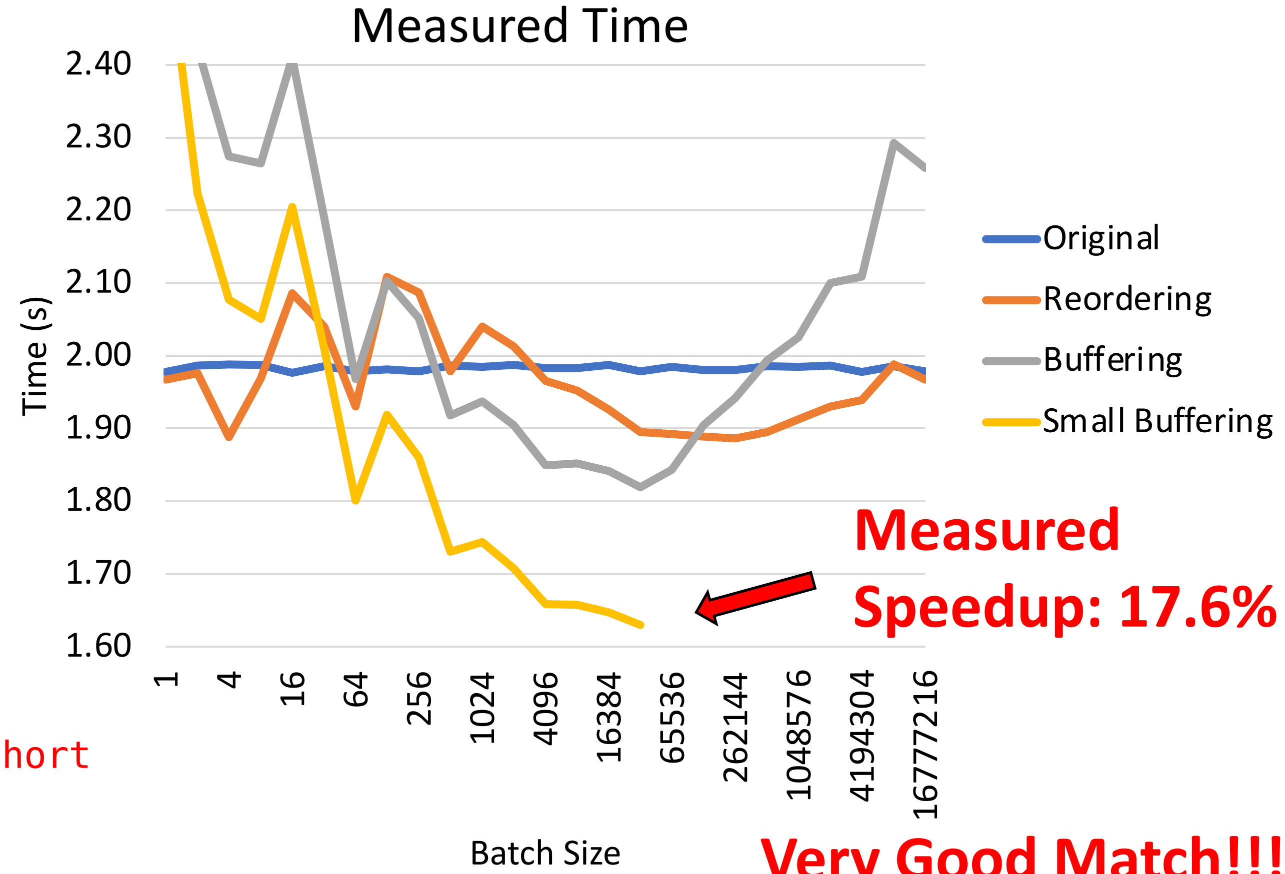
**Auxiliary Data:** $3N \times 8$ Bytes Read

**Matrix Data:** $\mathbf{27}N \times (8+2)$ Bytes Read → 17% B/W Saving

**Vector Data:** $N \times 8$ Bytes Read → Irregular Access Eliminated

$N \times 8$ Bytes Write

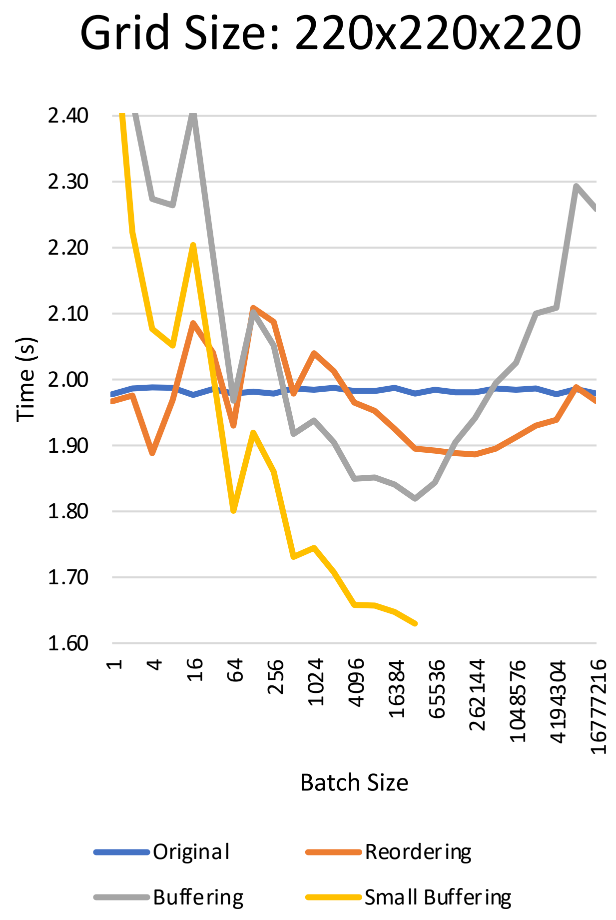
**Theoretical Speedup: 17%**



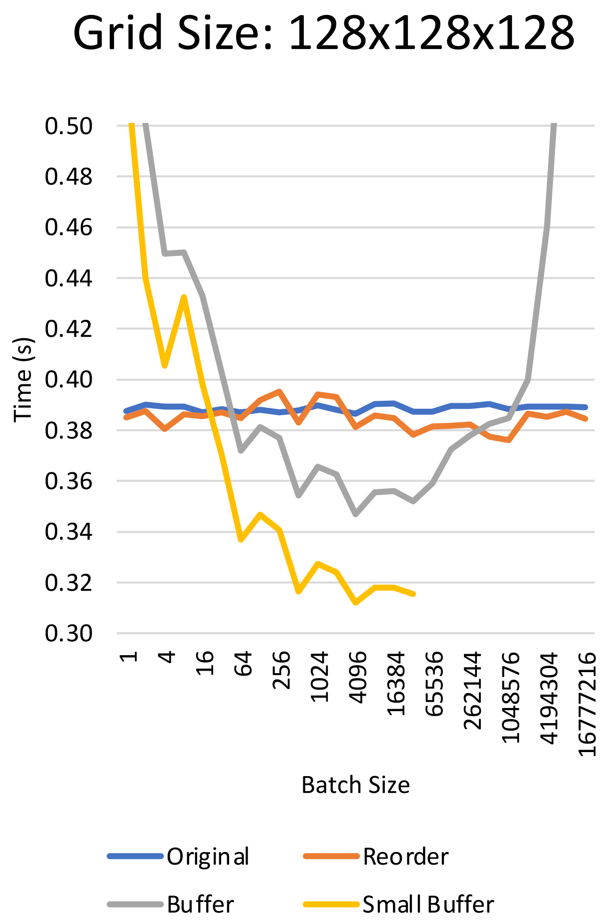

**Measured Speedup: 17.6%**

**Very Good Match!!!**

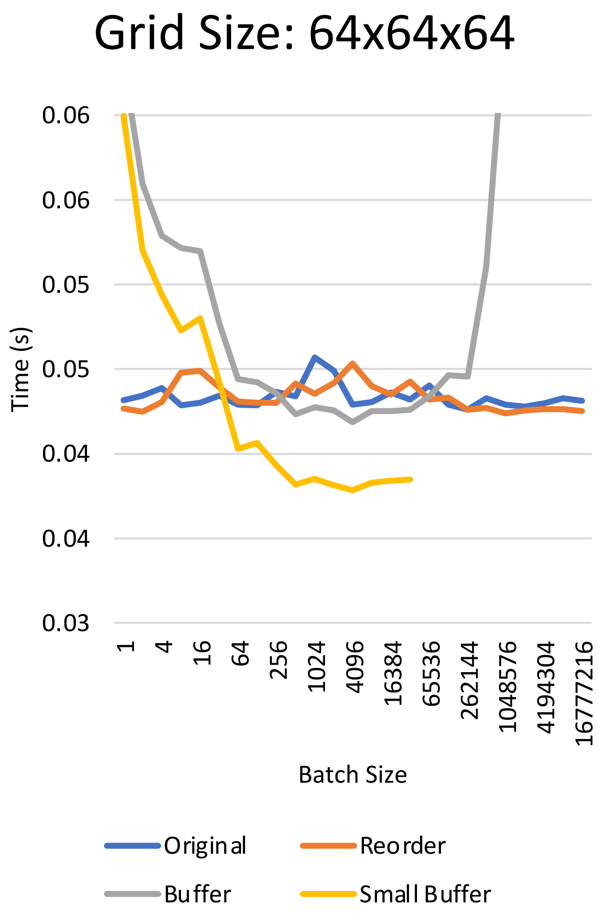
# Preliminary Results on Summit Node



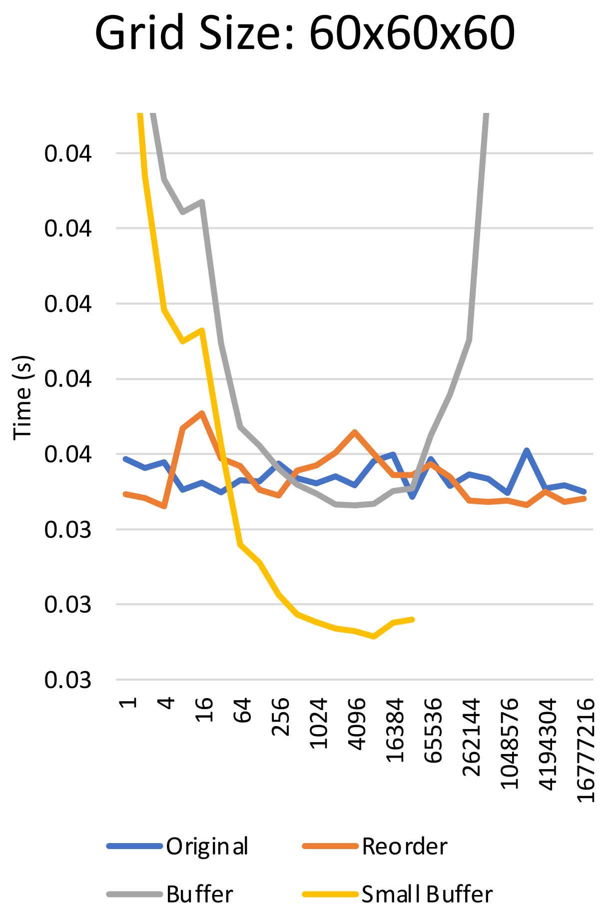

**Speedup: 17.6%**




**Speedup: 19.3%**




**Speedup: 15.7%**




**Speedup: 13.0%**

ECE ILLINOIS Department of Electrical and Computer Engineering
Coordinated Science Laboratory
IBM Research
C3SR center for cognitive computing systems research

# Verification: Solving Poisson Equation

$$\nabla^2 \phi = f \qquad \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = f$$

**Taylor Series Expansion at $\phi_i$**

$$\phi_{i-1} = \phi_i - \frac{\partial \phi_i}{\partial x} h + \frac{\partial^2 \phi_i}{\partial x^2} \frac{h^2}{2!} - \frac{\partial^3 \phi_i}{\partial x^3} \frac{h^3}{3!} + O(h^4)$$

$$\phi_{i+1} = \phi_i + \frac{\partial \phi_i}{\partial x} h + \frac{\partial^2 \phi_i}{\partial x^2} \frac{h^2}{2!} + \frac{\partial^3 \phi_i}{\partial x^3} \frac{h^3}{3!} + O(h^4)$$

---

$$\phi_{i-1} + \phi_{i+1} = 2\phi_i + \frac{\partial^2 \phi_i}{\partial x^2} h^2 + O(h^4)$$

$$\frac{\partial^2 \phi}{\partial x^2} = \frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{h^2} + O(h^2)$$

$$\frac{\partial^2 \phi}{\partial y^2} = \frac{\phi_{j-1} - 2\phi_j + \phi_{j+1}}{h^2} + O(h^2)$$

$$\frac{\partial^2 \phi}{\partial z^2} = \frac{\phi_{k-1} - 2\phi_k + \phi_{k+1}}{h^2} + O(h^2)$$

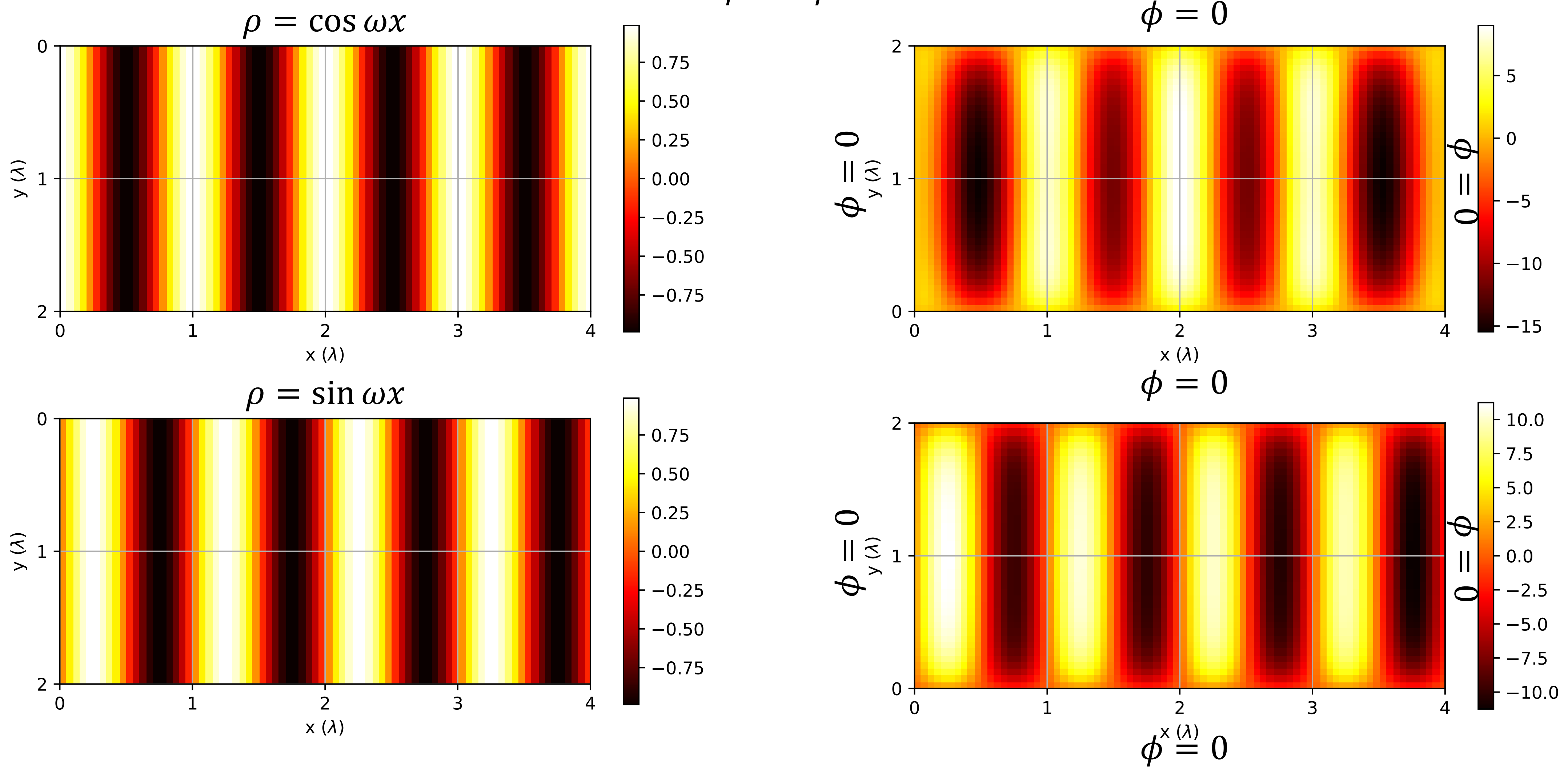
**3-point Equation:** $\delta_x^2 \phi = \dfrac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{h^2} = f_i + \mathcal{O}(h^2)$

**5-point Equation:** $[\delta_x^2 + \delta_y^2]\phi = \dfrac{\phi_{i-1} + \phi_{j-1} - 4\phi_{i,j} + \phi_{i+1} + \phi_{j+1}}{h^2} = f_{i,j} + \mathcal{O}(h^2)$

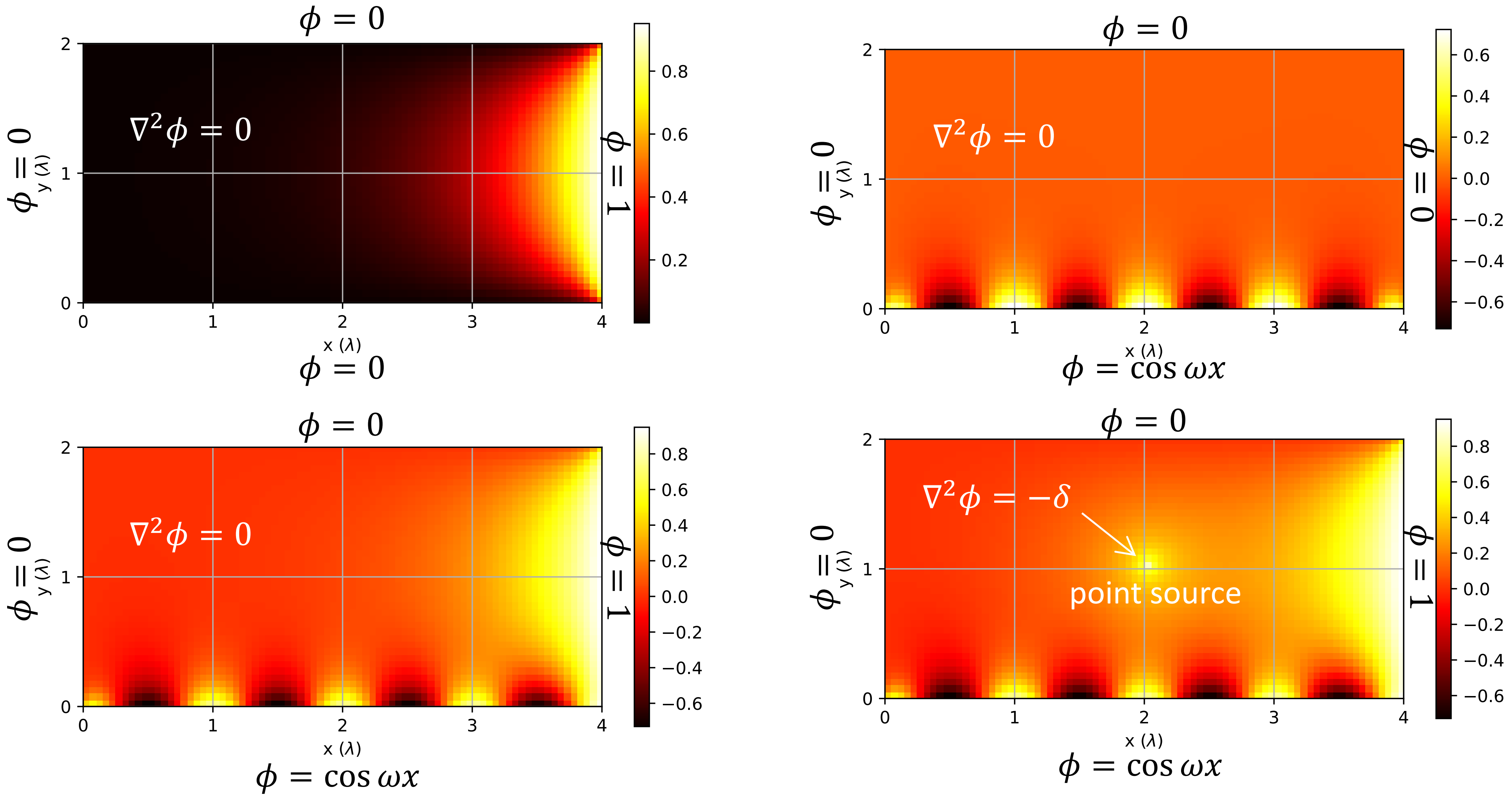
**7-point Equation:** $[\delta_x^2 + \delta_y^2 + \delta_z^2]\phi = \dfrac{\phi_{i-1} + \phi_{j-1} + \phi_{k-1} - 6\phi_{i,j,k} + \phi_{i+1} + \phi_{j+1} + \phi_{k+1}}{h^2} = f_{i,j,k} + \mathcal{O}(h^2)$

# Verification: Solving Poisson Equation with Dirichlet Boundary Conditions

$$\nabla^2 \phi = -\rho$$



# Verification: Solving Poisson Problem with Dirichlet Boundary Conditions




ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

C³SR center for cognitive computing systems research

## Verification $\nabla\phi = f$

**7 Points:** $\nabla_7^2\phi = \left[\delta_x^2 + \delta_y^2 + \delta_z^2\right]\phi = f + \mathcal{O}(h^2)$

**19 Points:** $\nabla_{19}^2\phi = \left[\nabla_7^2 + \frac{h^2}{6}\left(\delta_x^2\delta_y^2 + \delta_y^2\delta_z^2 + \delta_x^2\delta_z^2\right)\right]\phi = f + \frac{h^2}{12}\nabla_7^2 f + \mathcal{O}(h^4)$

**27 Points:** $\nabla_{27}^2\phi = \left[\nabla_{19}^2 + \frac{h^4}{30}\left(\delta_x^2 + \delta_y^2 + \delta_z^2\right)\right]\phi = f + \frac{h^2}{12}\nabla^2 f + \frac{h^4}{360}\nabla^4 f + \frac{h^4}{180}\left(\frac{\partial^4 f}{\partial x^2 \partial y^2} + \frac{\partial^4 f}{\partial y^2 \partial z^2} + \frac{\partial^4 f}{\partial x^2 \partial z^2}\right) + \mathcal{O}(h^6)$

**Function 1:**

$\phi = xyz \quad f = 0$

**Function 2:**

$\phi = x^2y^2z^2 \quad f = 2y^2z^2 + 2x^2z^2 + 2x^2y^2$

**Function 3:**

$\phi = x^3y^3z^3 \quad f = 6xy^3z^3 + 6x^3yz^3 + 6x^3y^3z$

**Function 4:**

$\phi = \sin(x+y+z)\cos(x+y+z) \quad f = -6\sin\left(2(x+y+z)\right)$

$$\nabla^2 f = 72\sin\left(2(x+y+z)\right)$$

$$\nabla^4 f = -864\sin\left(2(x+y+z)\right)$$

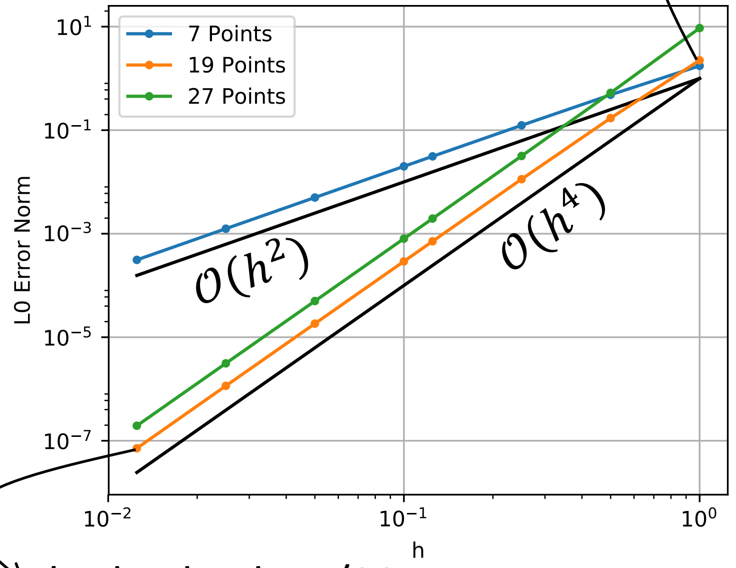
$$\frac{\partial^4 f}{\partial x^4} = \frac{\partial^4 f}{\partial y^4} = \frac{\partial^4 f}{\partial y^4} = \frac{\partial^4 f}{\partial x^2 y^2} = \frac{\partial^4 f}{\partial x^2 z^2} = \frac{\partial^4 f}{\partial y^2 z^2} = -96\sin\left(2(x+y+z)\right)$$

# Verification: Forward Problem

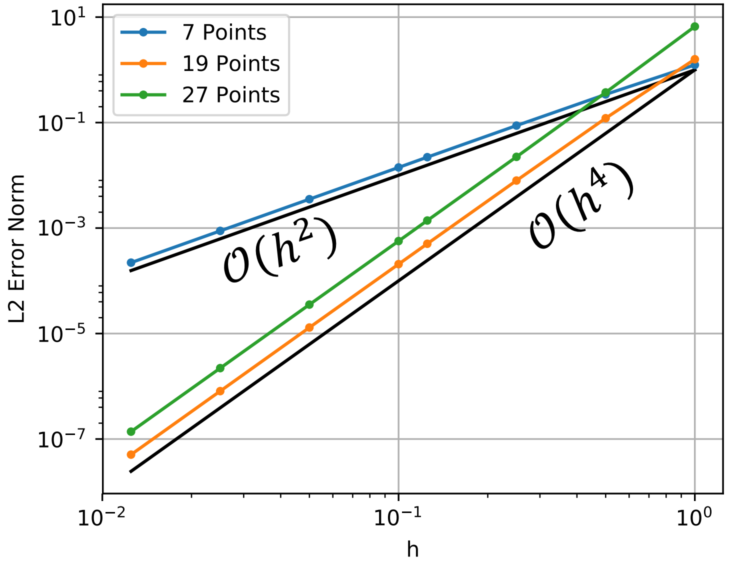
$$\phi = \sin(x+y+z)\cos(x+y+z) \Longrightarrow \nabla^2\phi = -6\sin\big(2(x+y+z)\big)$$

h=dx=dy=dz=1
3x3x3 grid

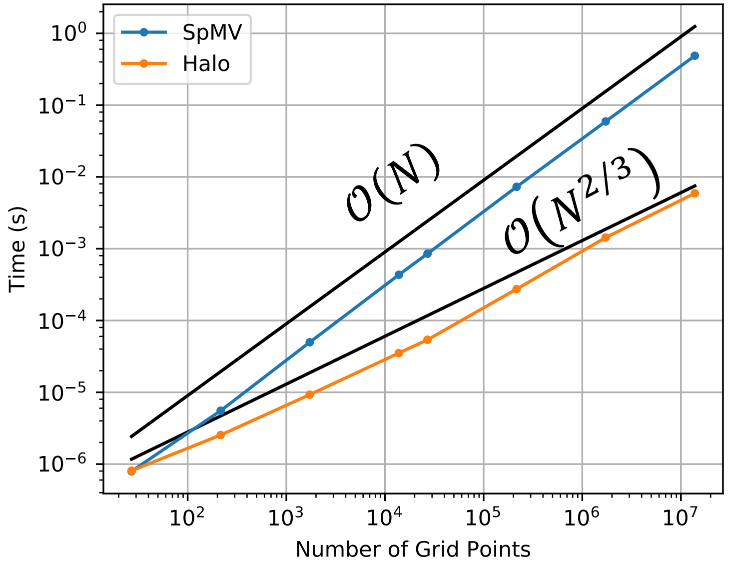
$L0 = \max(|e_i|)$




$$L2 = \left(\frac{\sum e_i^2}{N}\right)^{0.5}$$




Measured Time



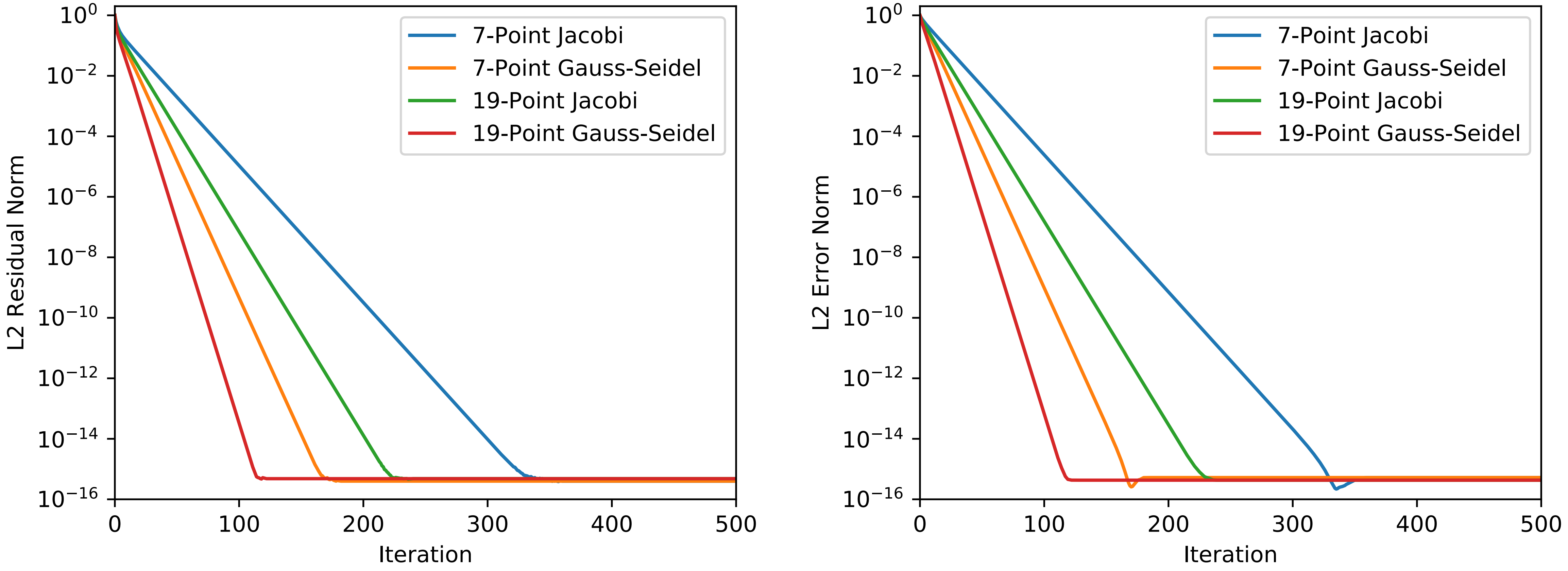

h=dx=dy=dz=1/80
240x240x240 grid

See 7-point, 19-point, and 27-point derivations at: W. F. Spotz and G. F. Carey, “A high-order compact formulation for the 3D Poisson equation,” Numerical Methods for Partial Differential Equations, vol. 12, pp. 235-243, 1996.

# Verification: Inverse Problem

$$\nabla^2\phi = -6\sin\big(2(x+y+z)\big) \Rightarrow \phi = \sin(x+y+z)\cos(x+y+z)$$

6x6x6
h=0.0001

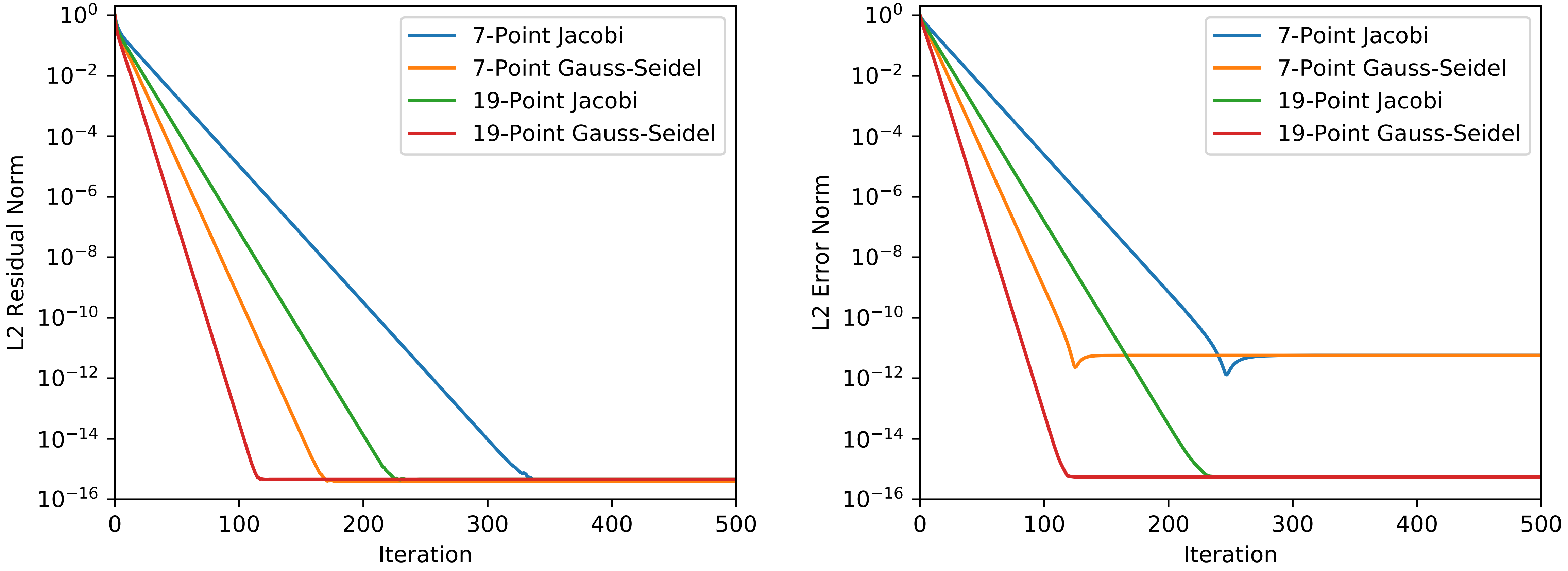


# Verification: Inverse Problem

$$\nabla^2 \phi = -6\sin\big(2(x+y+z)\big) \Longrightarrow \phi = \sin(x+y+z)\cos(x+y+z)$$

6x6x6

h=0.001

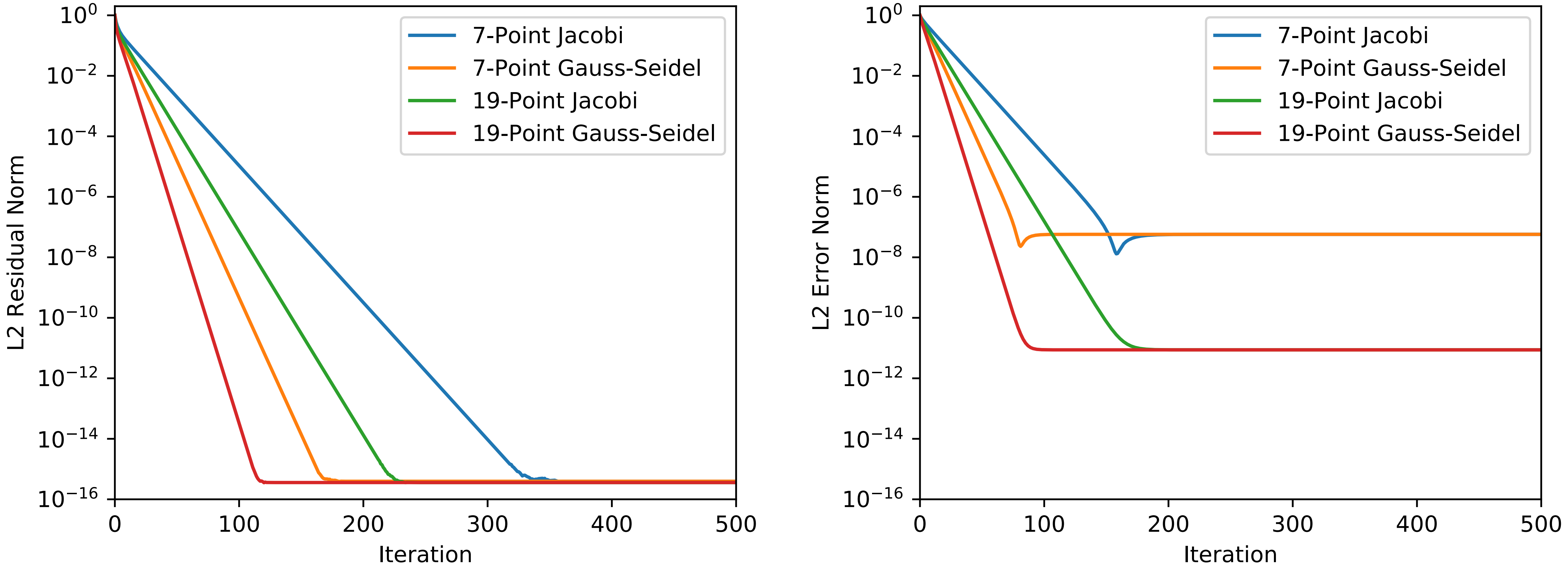


# Verification: Inverse Problem

$$\nabla^2 \phi = -6\sin\big(2(x+y+z)\big) \quad \Rightarrow \quad \phi = \sin(x+y+z)\cos(x+y+z)$$

6x6x6

h=0.01

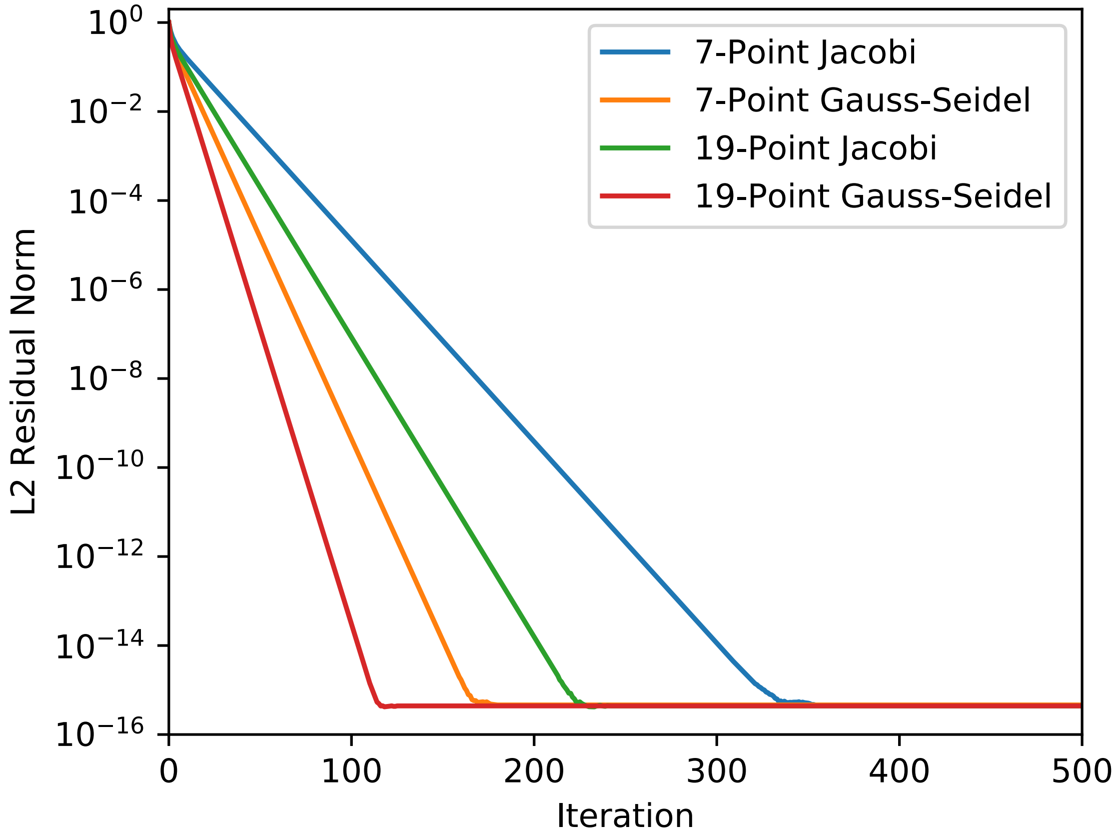


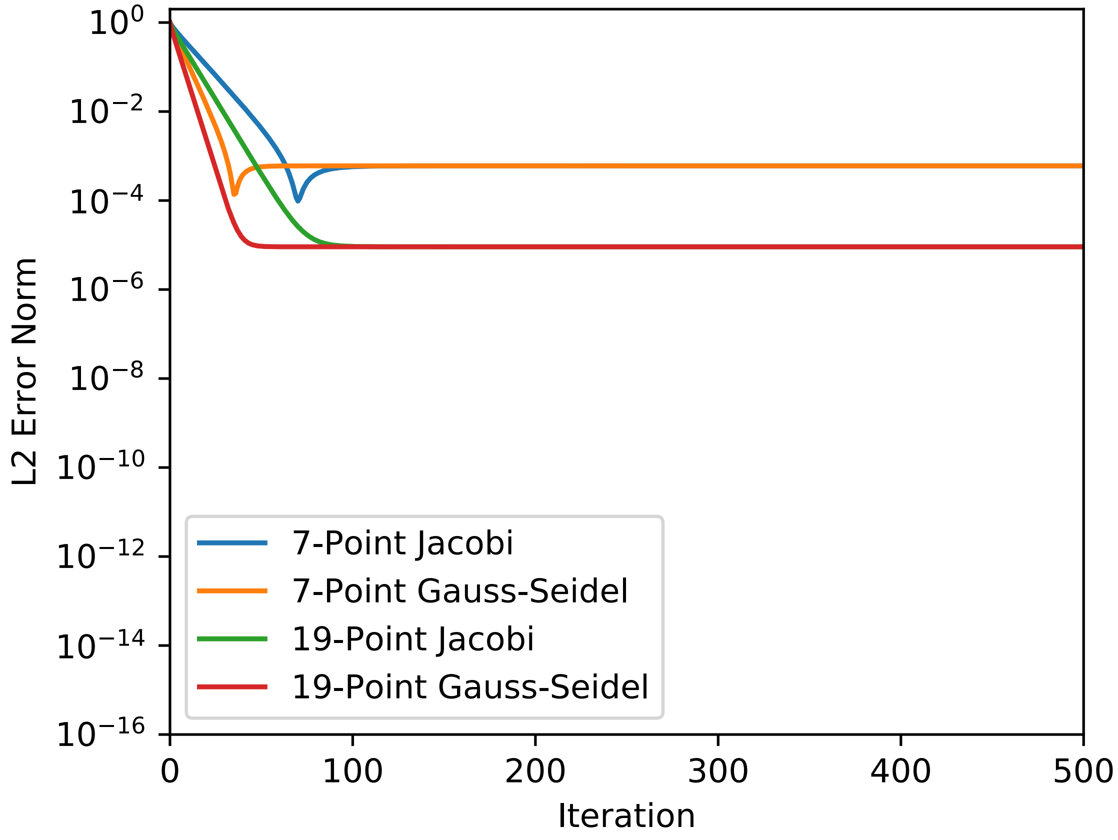
# Verification: Inverse Problem

$$\nabla^2 \phi = -6\sin\big(2(x+y+z)\big) \quad \Rightarrow \quad \phi = \sin(x+y+z)\cos(x+y+z)$$

6x6x6

h=0.1



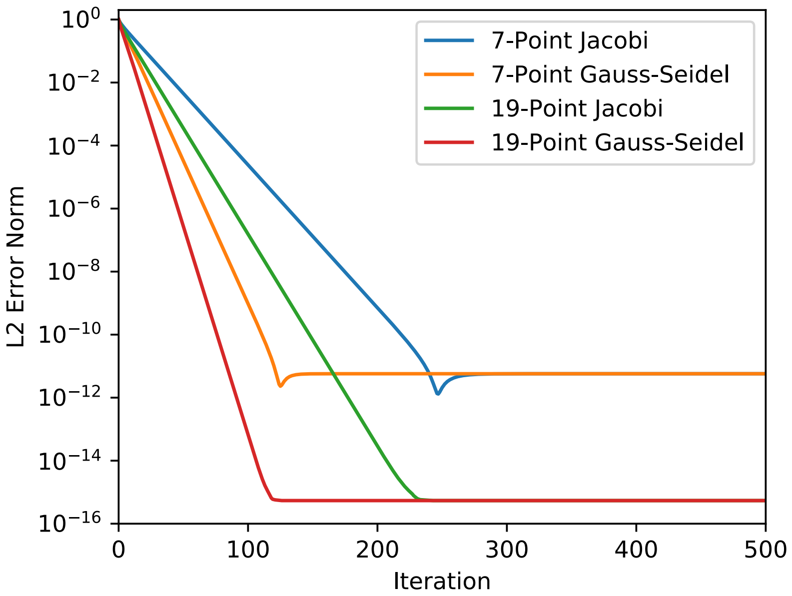
# Verification: Batch Reordering & Buffering Algorithms

$$\nabla^2\phi = -6\sin\big(2(x+y+z)\big) \Rightarrow \phi = \sin(x+y+z)\cos(x+y+z)$$

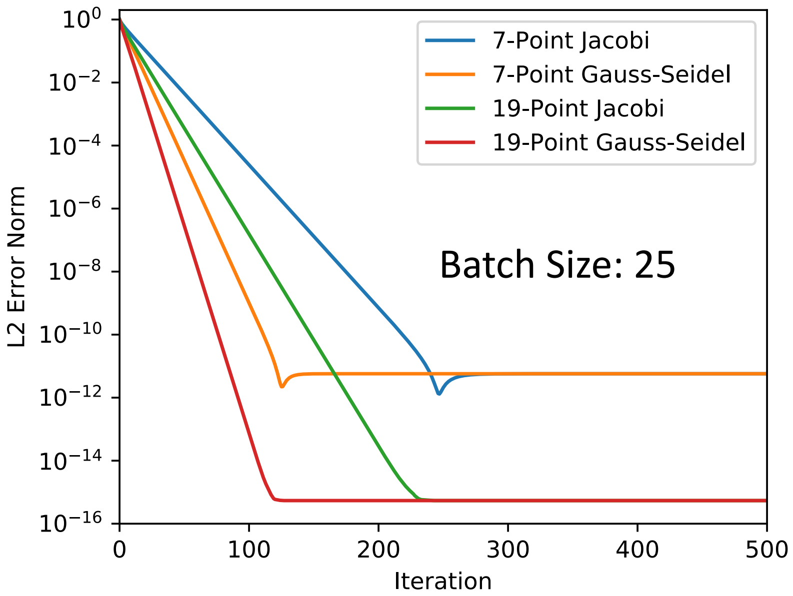
6x6x6

h=0.001

## Natural Ordering




## Batch Reordering



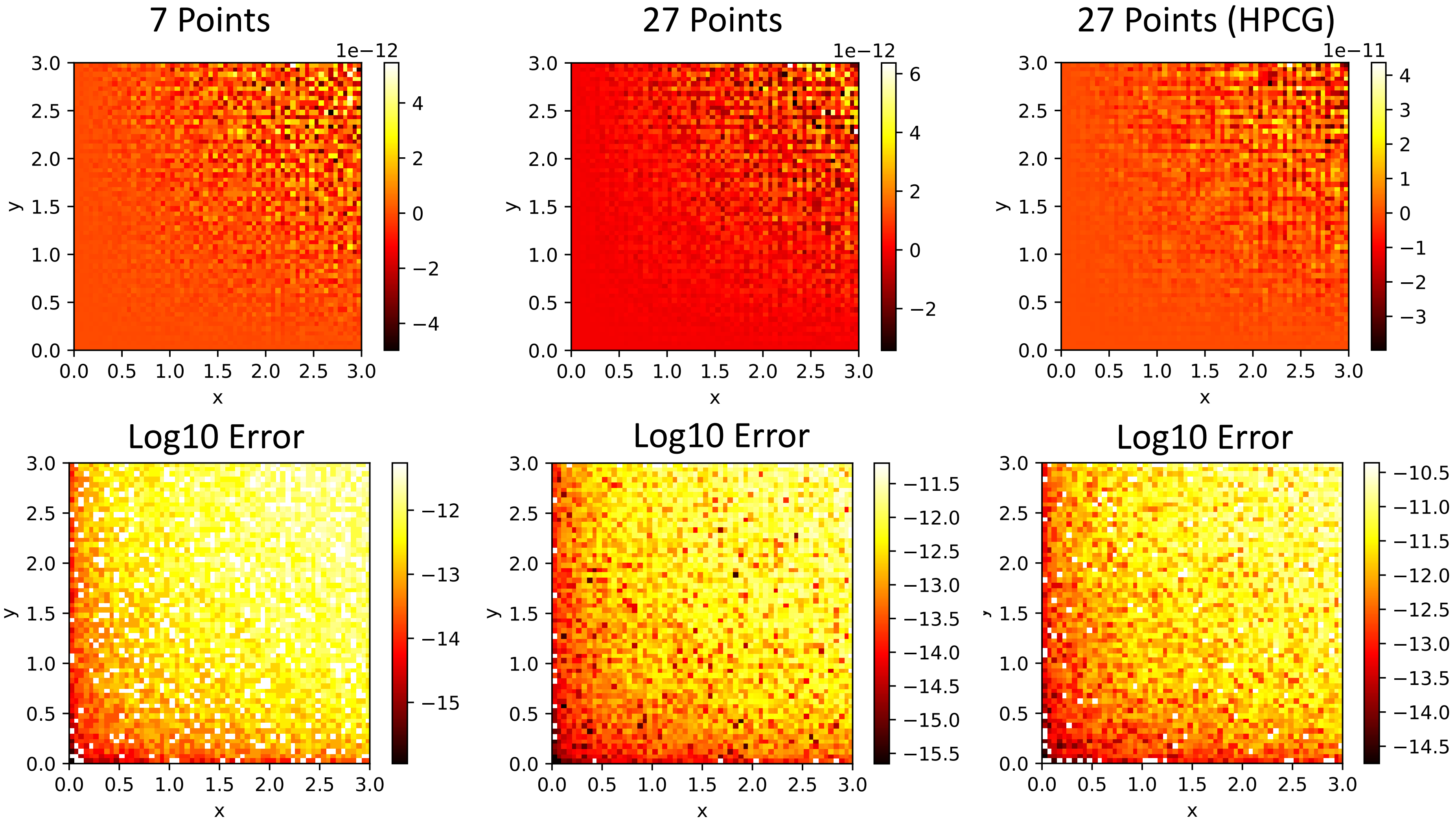
# Verification: Forward Problem

Plots Shows z = 1.55

$$\phi = xyz$$

$$\nabla^2 \phi = 0$$

60x60x60, h = 0.05, 500 iter.
27-Point Stencil




max error: 1.45519152284e−11

max error: 1.45519152284e−11

max error: 1.01863406599e−10

ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

C3SR center for cognitive computing systems research

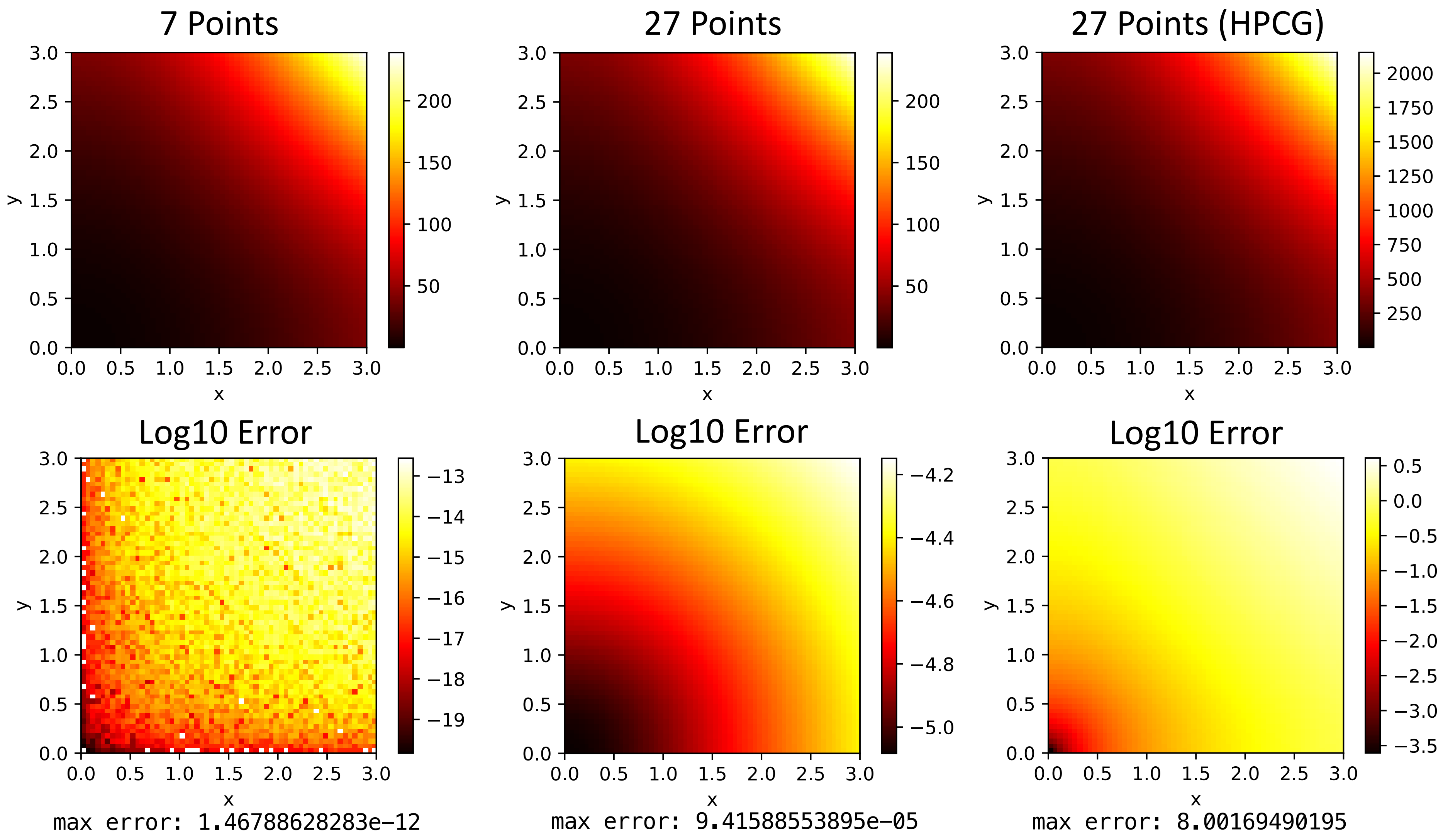
# Verification: Forward Problem

$$\phi = x^2y^2z^2$$

$$\nabla^2\phi = 2y^2z^2 + 2x^2z^2 + 2x^2y^2$$

60x60x60, h = 0.05, 500 iter.
27-Point Stencil

Plots Shows z = 1.55




ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

C3SR center for cognitive computing systems research

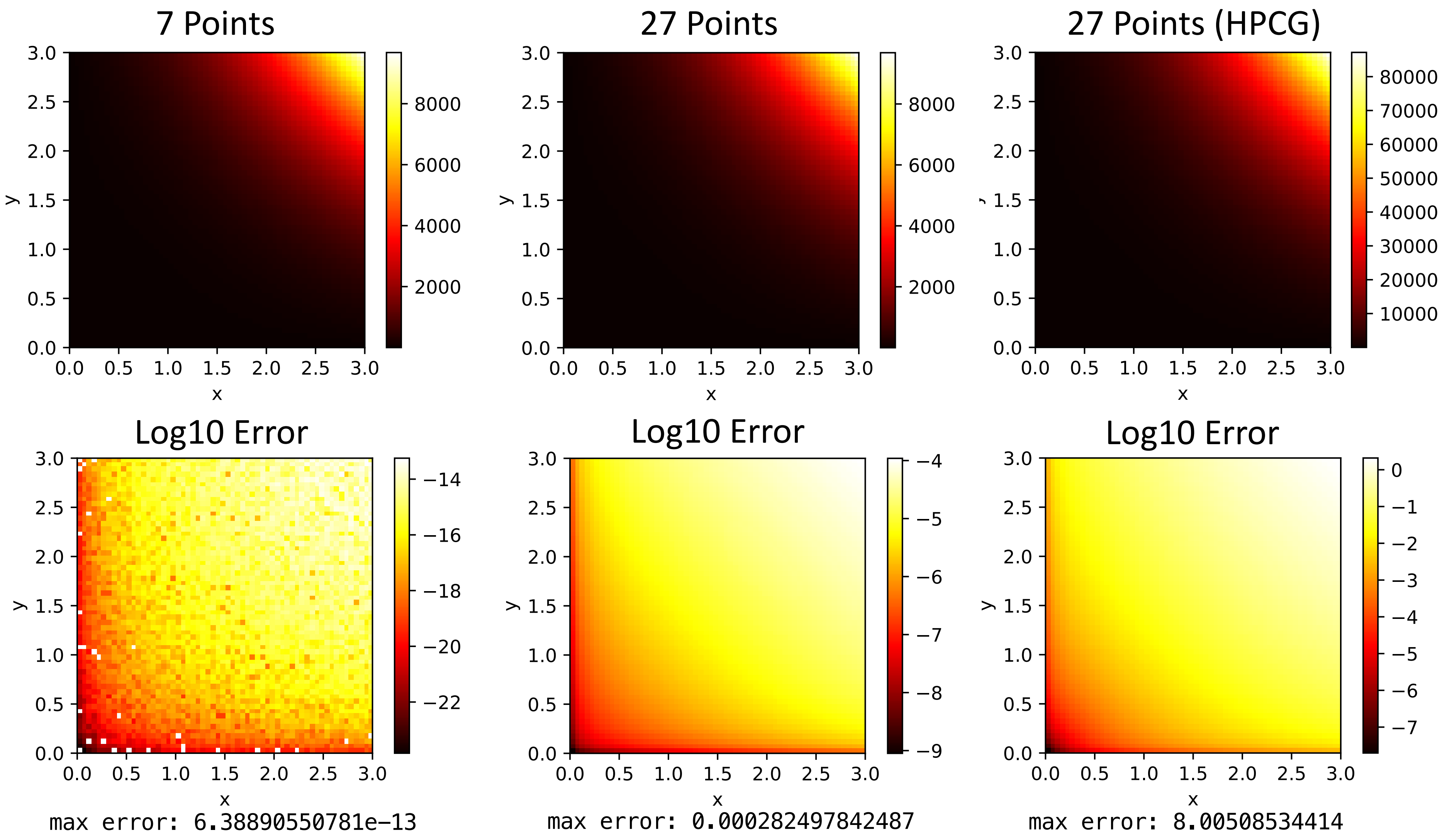
# Verification: Forward Problem

Plots Shows z = 1.55

$$\phi = x^3y^3z^3$$

$$\nabla^2\phi = 6xy^3z^3 + 6x^3yz^3 + 6x^3y^3z$$

60x60x60, h = 0.05, 500 iter.
27-Point Stencil




ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

C3SR center for cognitive computing systems research

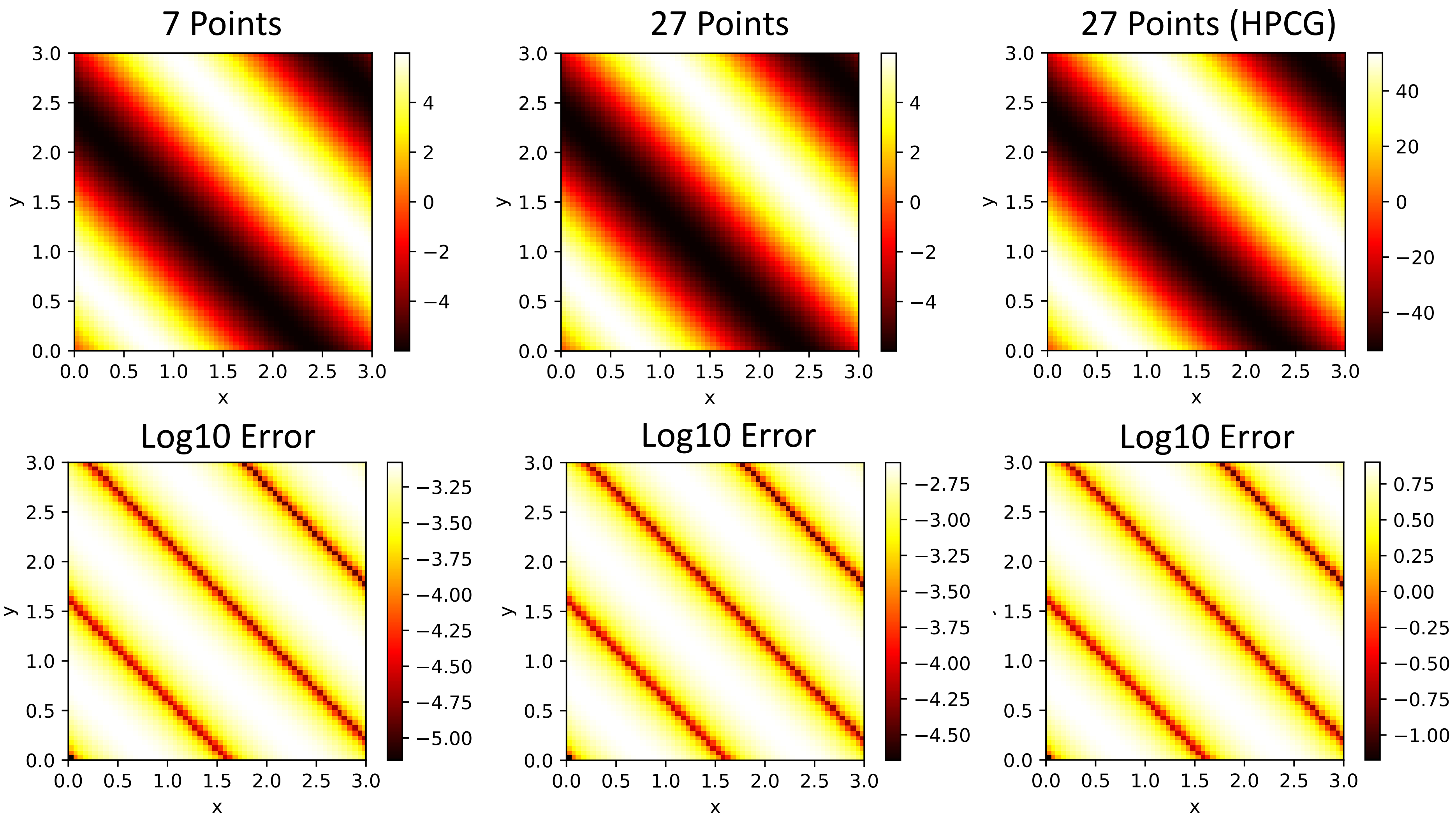
# Verification: Forward Problem

$$\phi = \sin(x+y+z)\cos(x+y+z)$$

$$\nabla^2\phi = -6\sin\big(2(x+y+z)\big)$$

60x60x60, h = 0.05, 500 iter.
27-Point Stencil

Plots Shows z = 1.55




max error: 0.000833055605171

max error: 0.0024958382399

max error: 7.96258571217

ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

C3SR center for cognitive computing systems research

# Least Squares: Gradient-Descent Method

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

**Finding Gradient:**

$$f(x+d) = f(x) + d^T(Ax-b) + \frac{1}{2}d^T Ad$$

$$= f(x) + d^T \nabla f(x) + O(d^2)$$

$$\boxed{\nabla f(x) = (Ax-b) = r}$$

**Finding Step Size:**

$$f(x+\alpha d) = f(x) + \alpha d^T r + \alpha^2 \frac{1}{2} d^T Ad$$

$$\frac{\partial f}{\partial \alpha} = d^T r + a d^T Ad = 0$$

$$\boxed{\alpha = -\frac{d^T r}{d^T Ad}}$$

**Gradient-Descent Step:**

$$d = \nabla(f) = r$$

$$x \leftarrow x - d\frac{d^T r}{d^T Ad}$$

$$r \leftarrow r - Ad\frac{d^T r}{d^T Ad}$$

⇦ Can be implemented with a single MVM

**Sanity Check: step direction has to be orthogonal to the next gradient**

$$\langle d, \nabla f(x+\alpha d)\rangle = d^T\left[Ax - Ad\frac{d^T r}{d^T Ad} - b\right] = d^T[Ax-b] - d^T r = 0$$

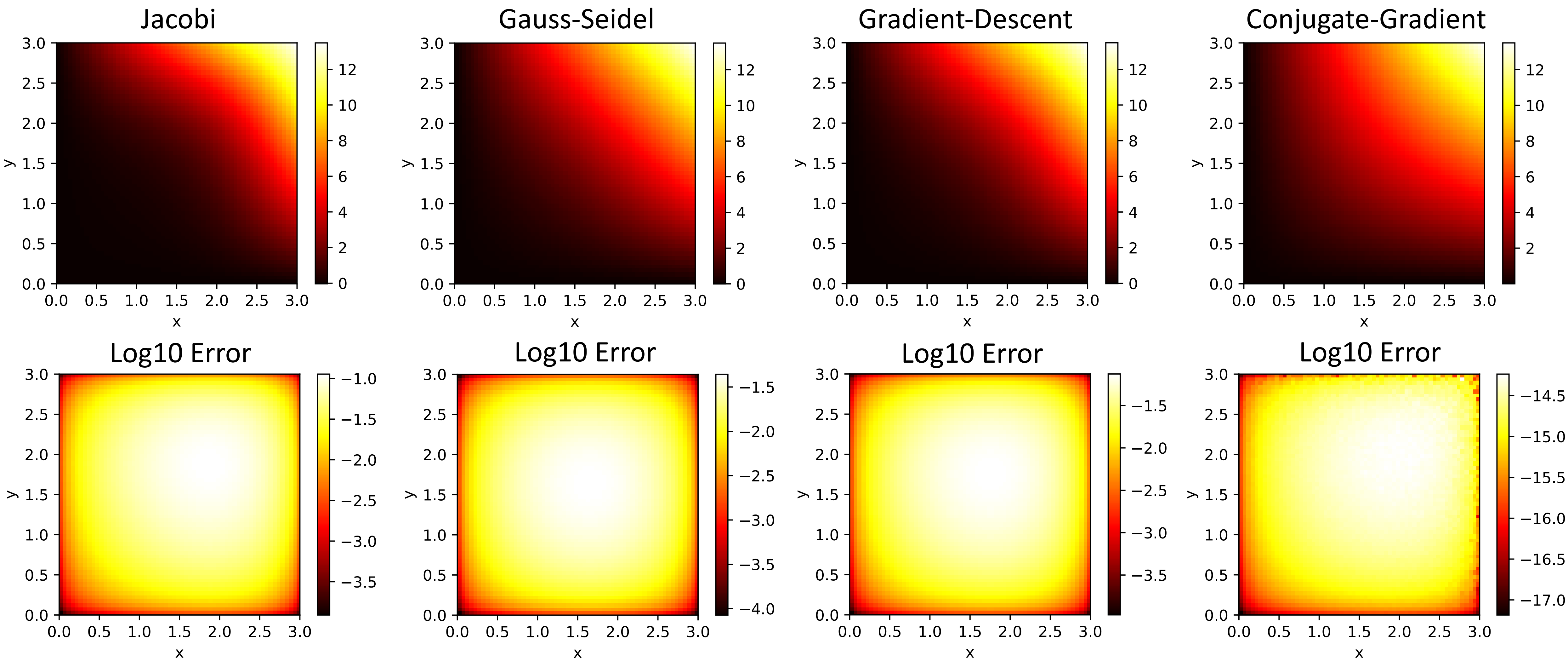
# Verification: Inverse Problem

Plots Shows z = 1.55

$$\nabla^2 \phi = 0$$

$$\phi = xyz$$

60x60x60, h = 0.05, 500 iter.
27-Point Stencil




max error: 0.121224243455

max error: 0.0442830865917

max error: 0.076706235115

max error: 7.0836745736e-15

ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

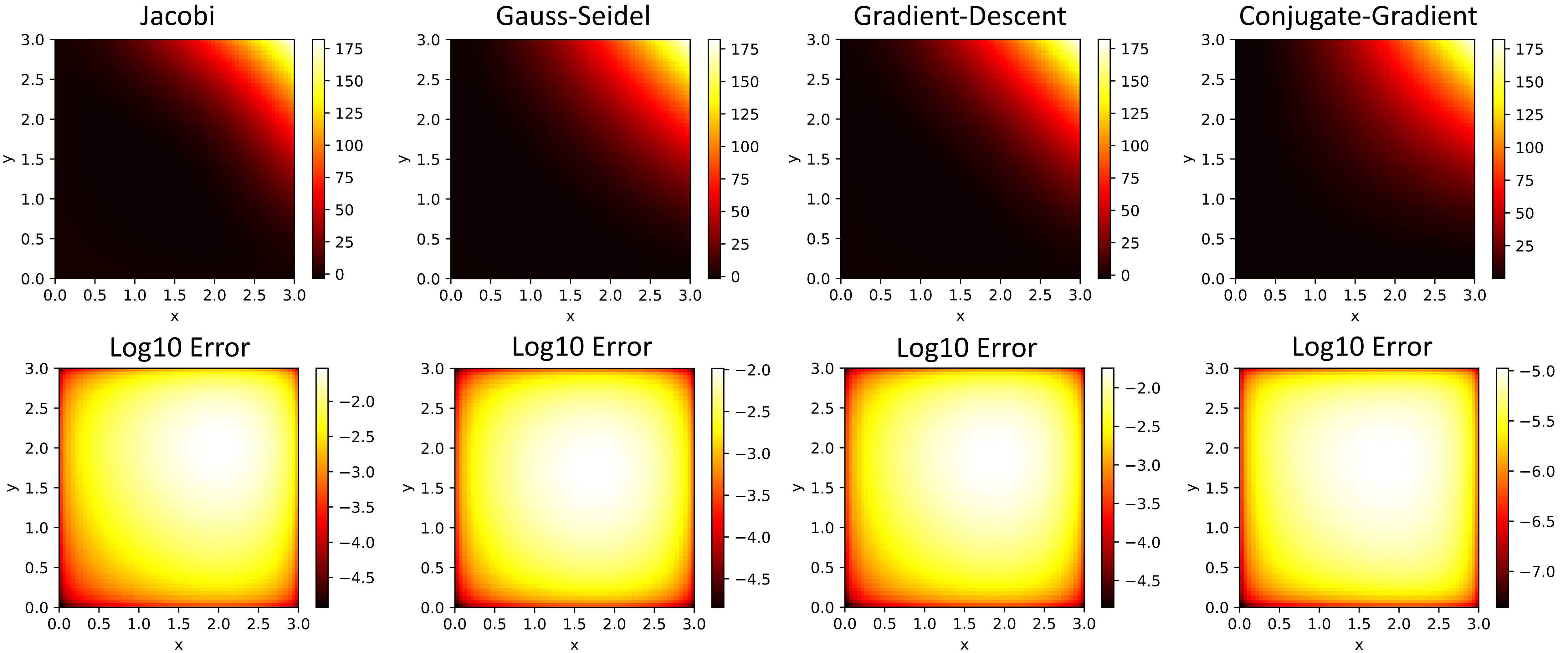
C3SR center for cognitive computing systems research

# Verification: Inverse Problem

$$\nabla^2\phi = 2y^2z^2 + 2x^2z^2 + 2x^2y^2$$
$$\phi = x^2y^2z^2$$

60x60x60, h = 0.05, 500 iter.
27-Point Stencil

Plots Shows z = 1.55




max error: 0.0347910309115

max error: 0.0106007486994

max error: 0.0195319652925

max error: 1.09441058053e-05

ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

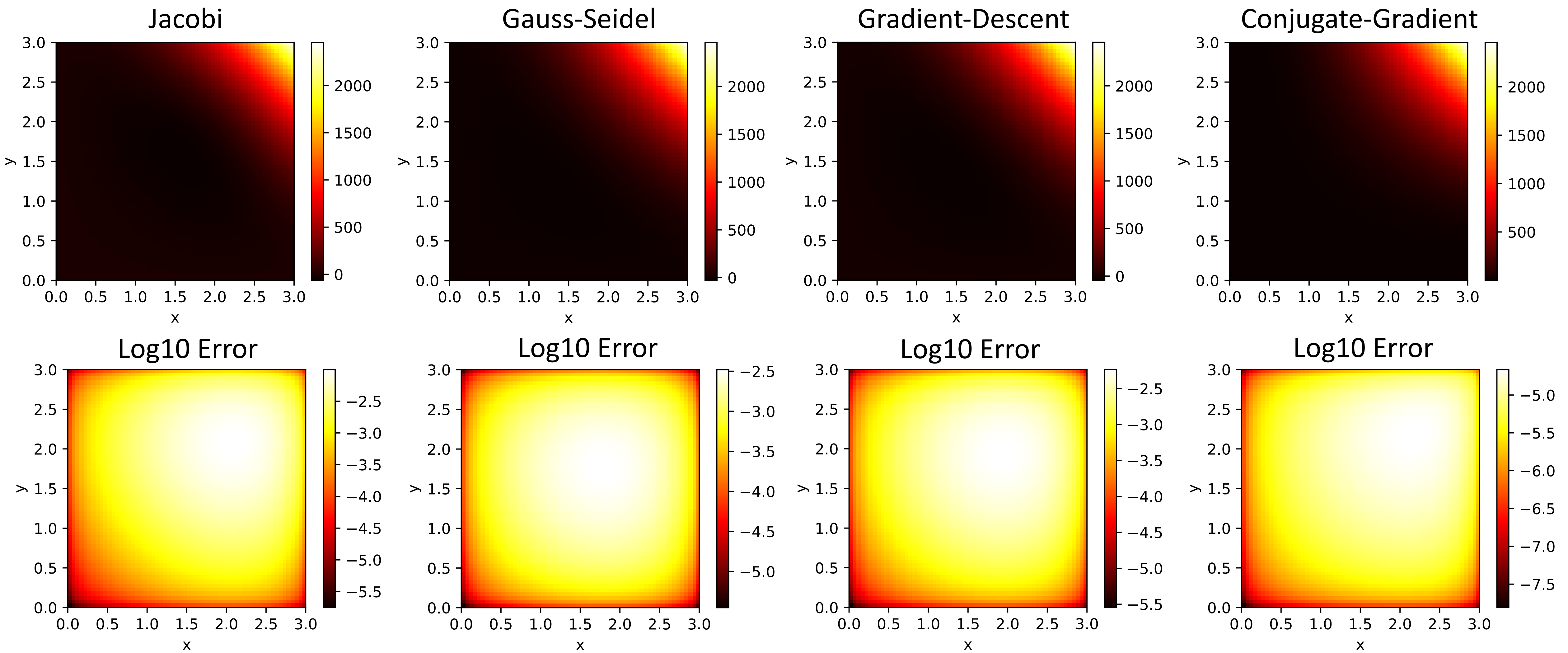
C3SR center for cognitive computing systems research

# Verification: Inverse Problem

$$\nabla^2 \phi = 6xy^3z^3 + 6x^3yz^3 + 6x^3y^3z$$
$$\phi = x^3y^3z^3$$

Plots Shows z = 1.55

60x60x60, h = 0.05, 500 iter.
27-Point Stencil




max error: 0.0128737509551

max error: 0.00342426882486

max error: 0.00661606913659

max error: 2.86034236171e-05

ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

IBM Research

C3SR center for cognitive computing systems research

# Verification: Inverse Problem

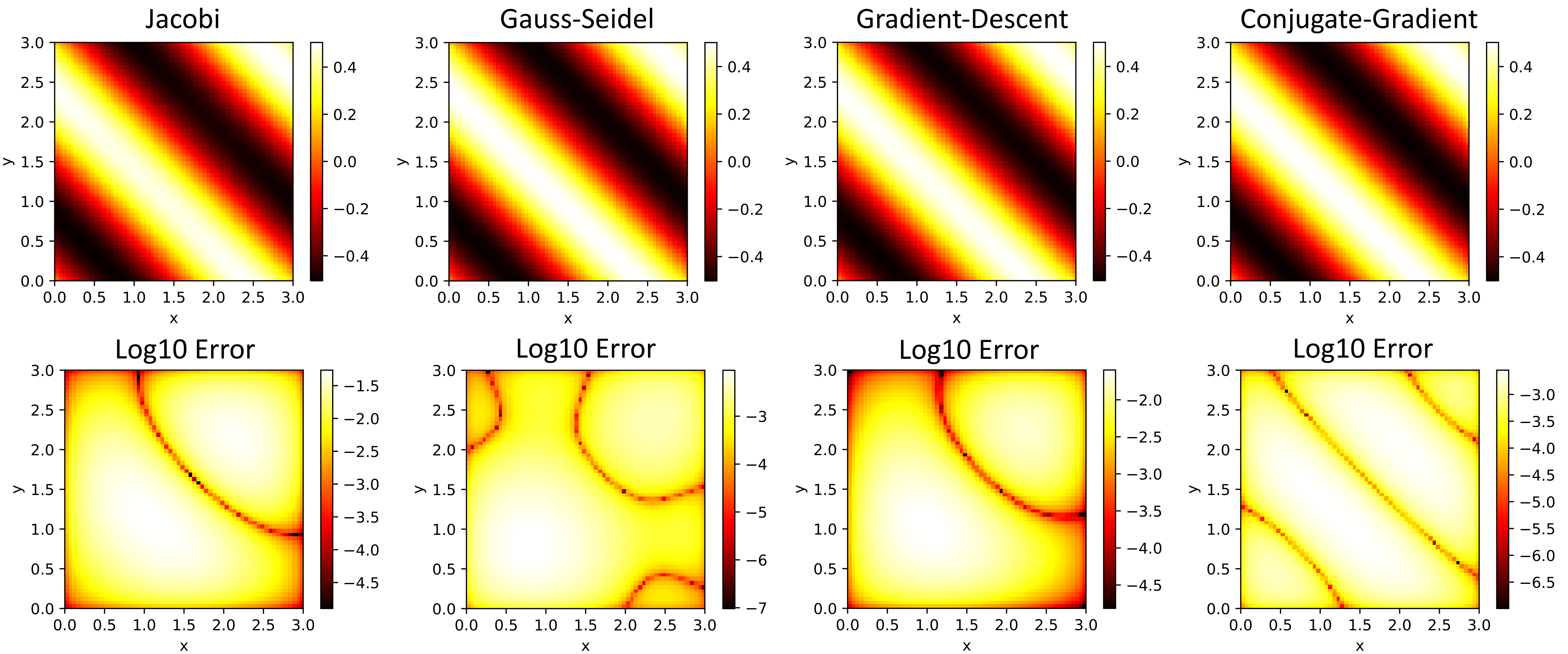
Plots Shows z = 1.55

$$\nabla^2\phi = -6\sin\big(2(x+y+z)\big)$$

$$\phi = \sin(x+y+z)\cos(x+y+z)$$

60x60x60, h = 0.05, 500 iter.

27-Point Stencil




max error: 0.0556809398082

max error: 0.0123857890091

max error: 0.0268054493562

max error: 0.00284205543708

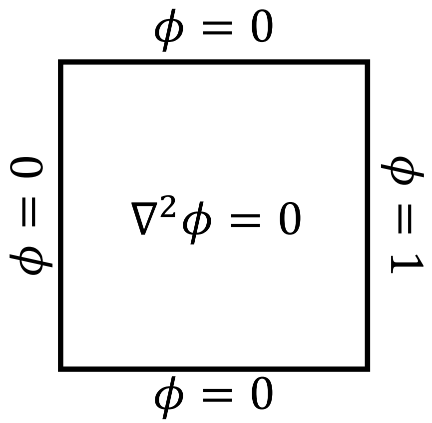
ECE ILLINOIS
Department of Electrical and Computer Engineering

Coordinated Science Laboratory

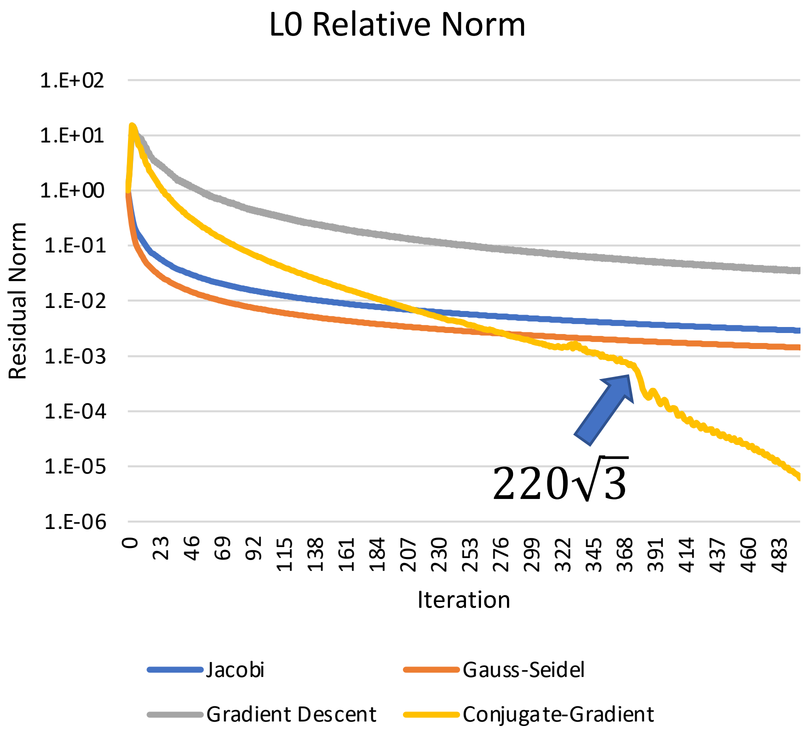
IBM Research

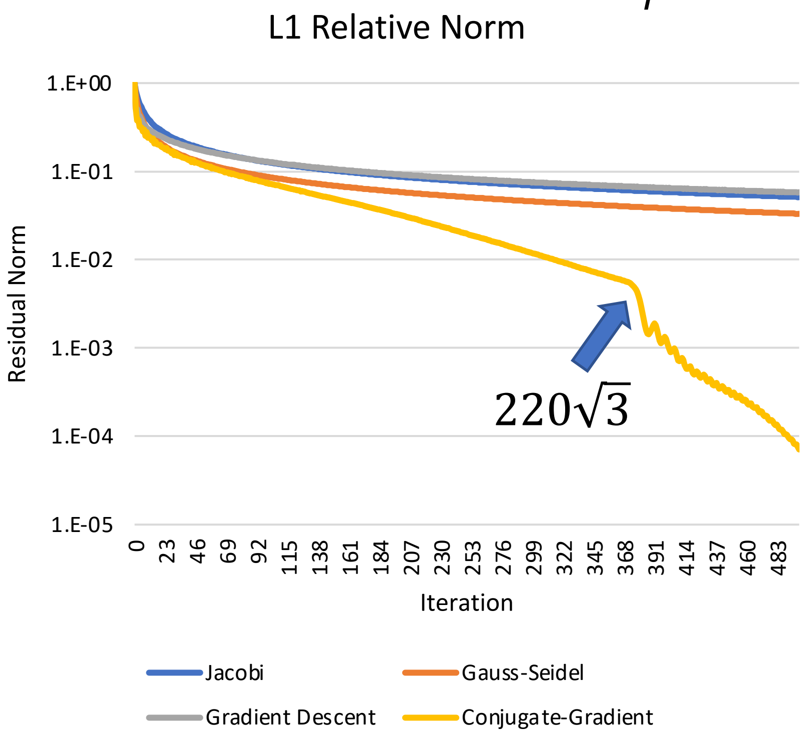
C3SR center for cognitive computing systems research

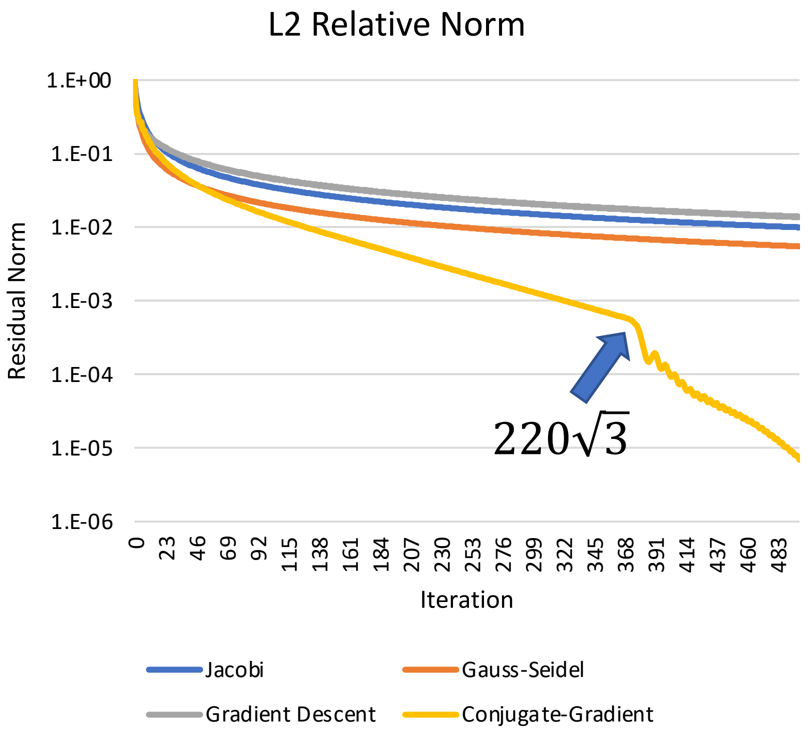
# Verification: Convergence




220x220x220
7-point equation

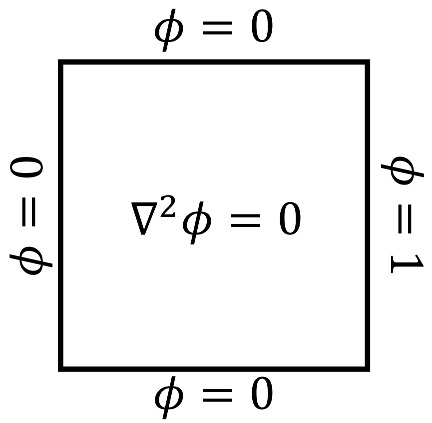


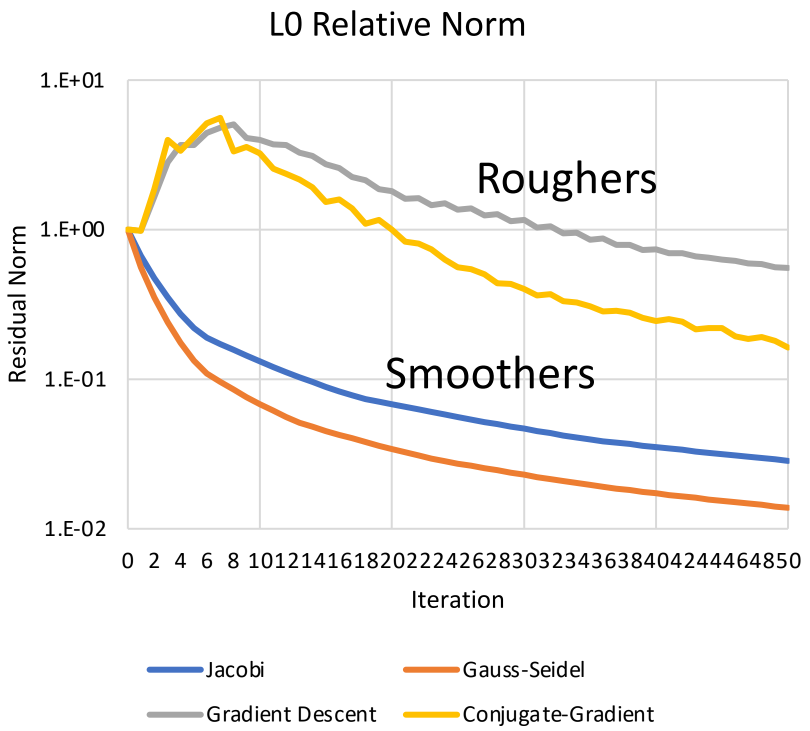



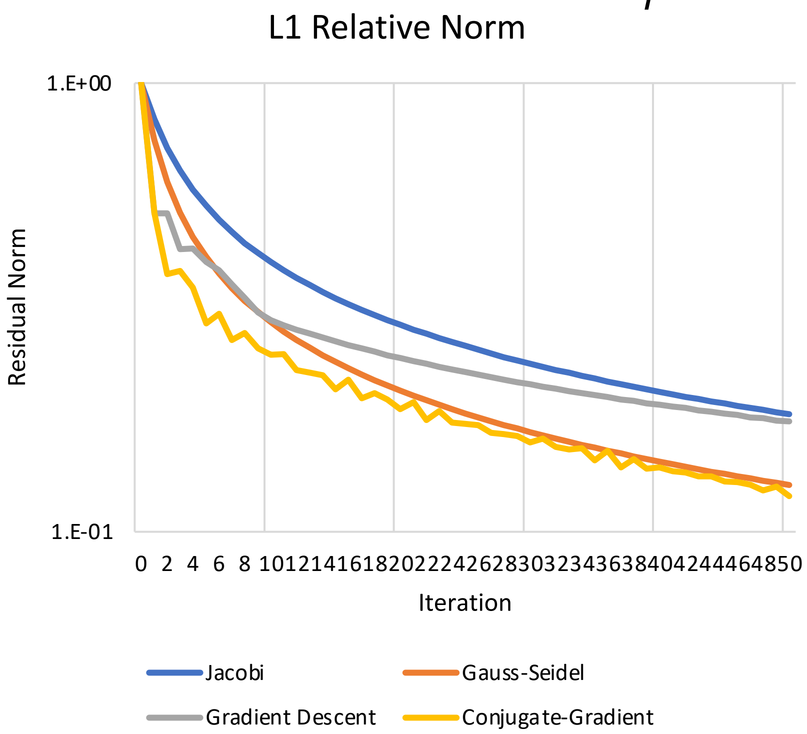
# Verification: Convergence

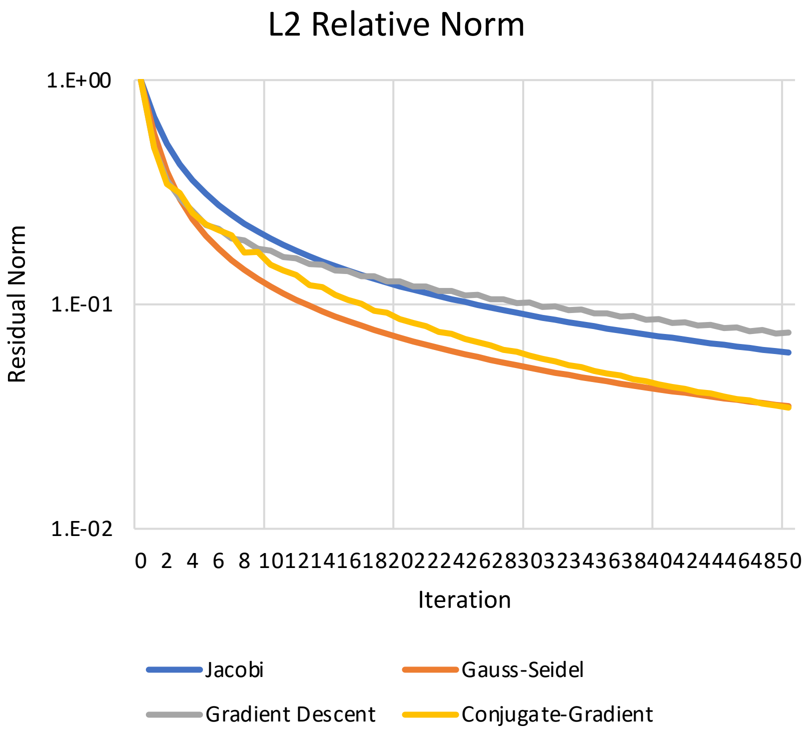
**ZOOMED!**



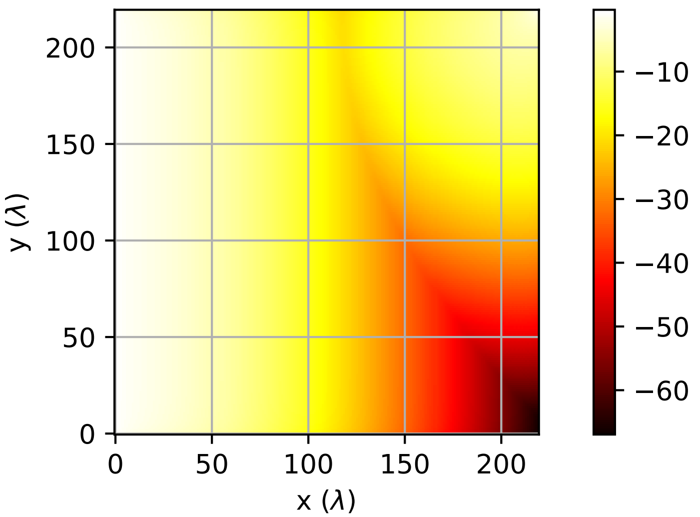

220x220x220
7-point equation



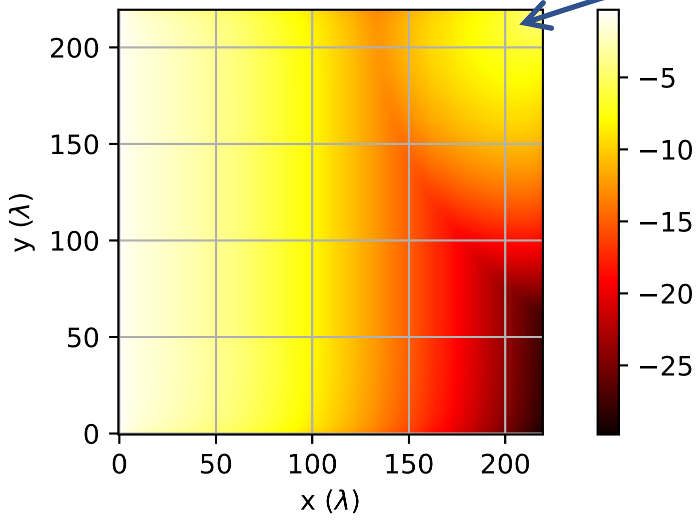



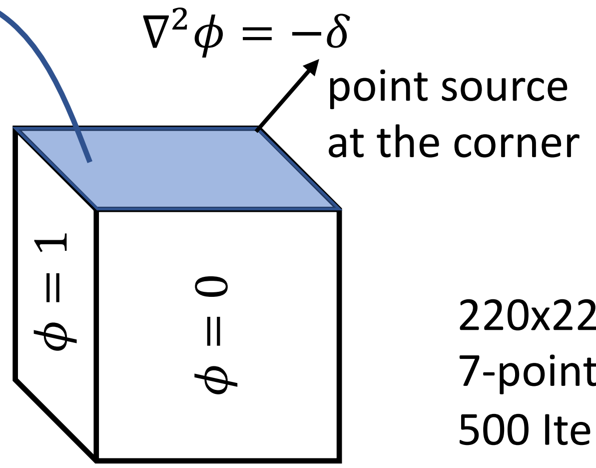
## Jacobi




## Gauss-Seidel




Top Layer

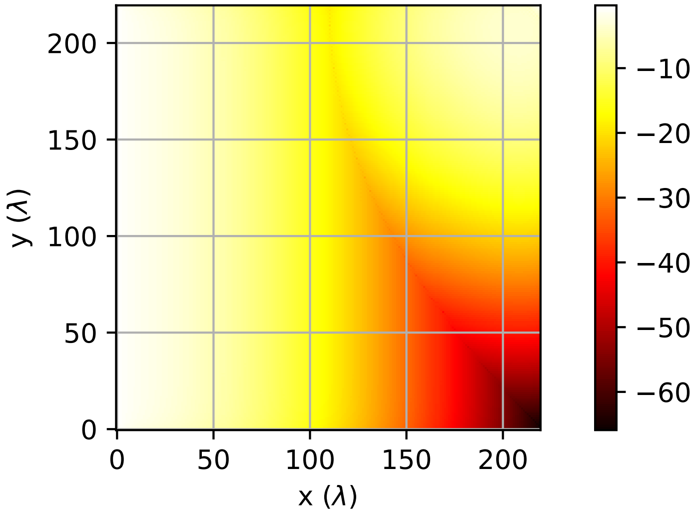



220x220x220

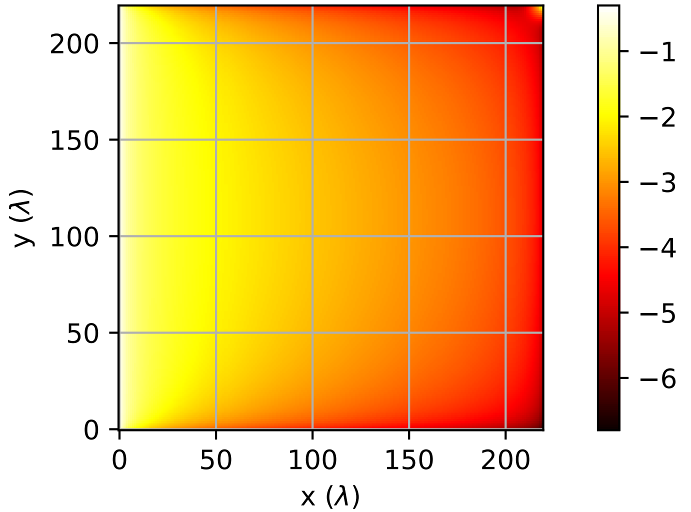
7-point equation

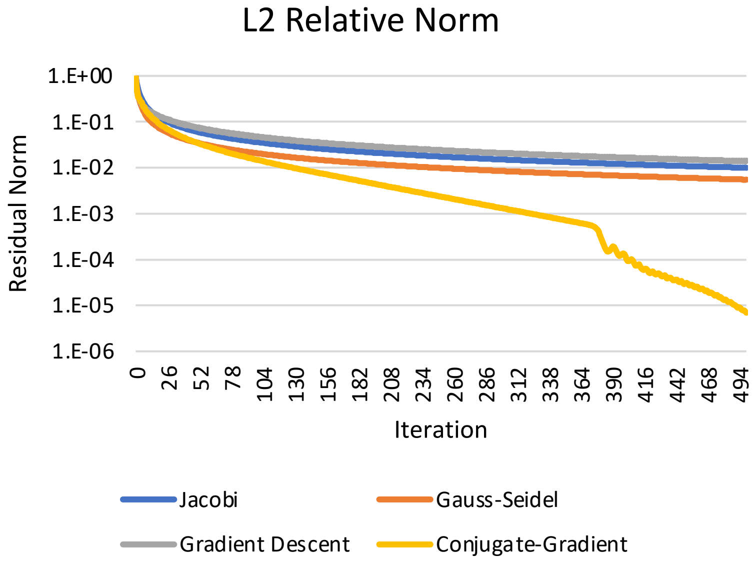
500 Iterations

## Gradient Descent



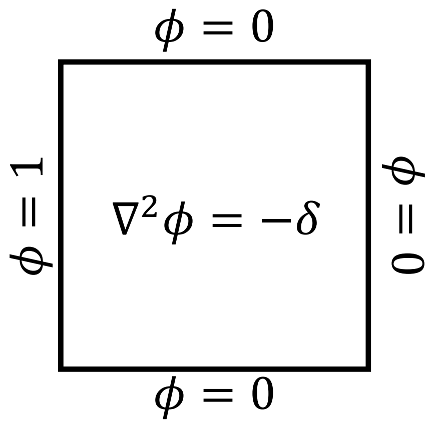

## Conjugate Gradient

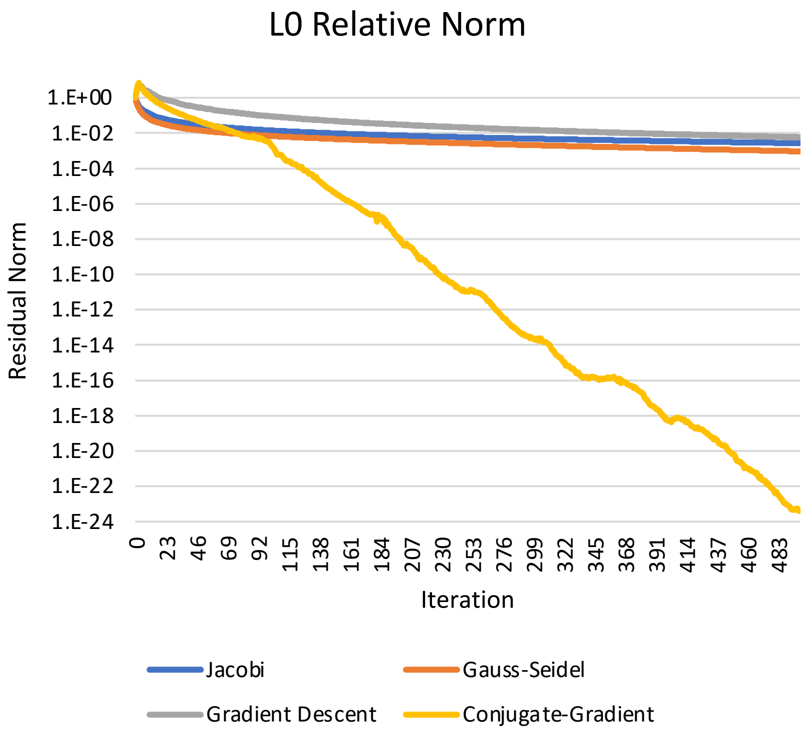


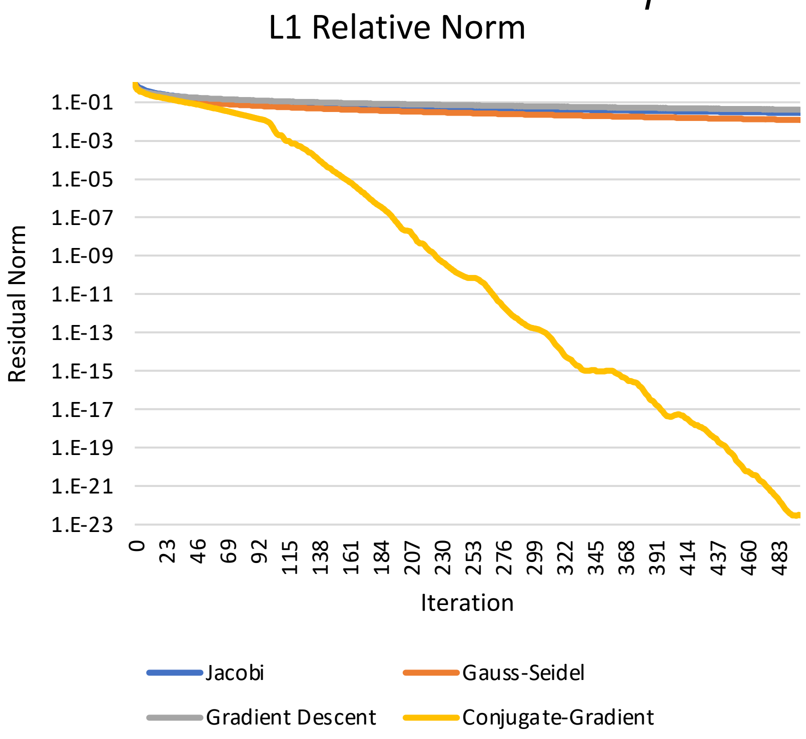



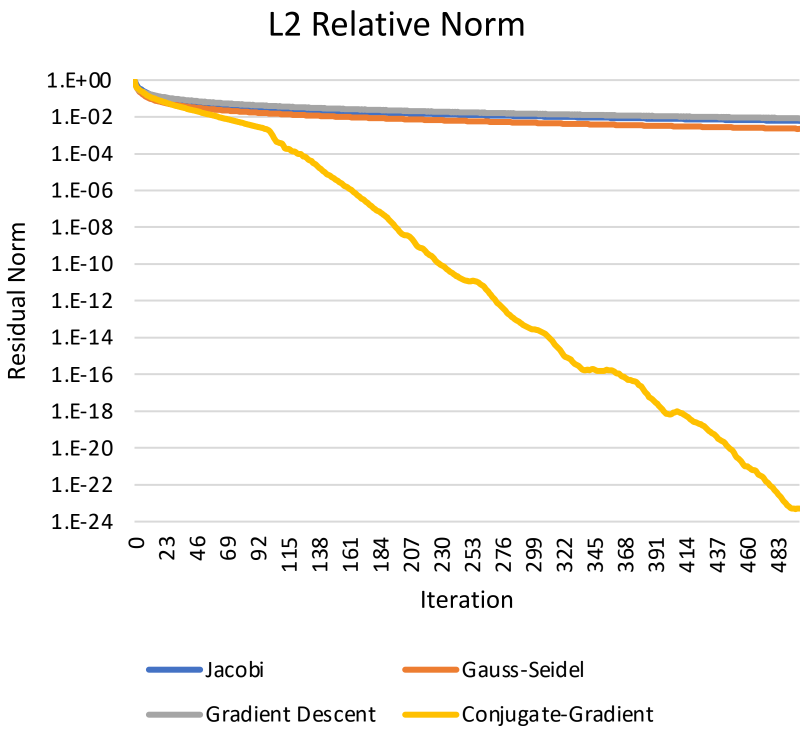
# Verification: Convergence

**60x60x60, h = 0.05**
**7-point equation**



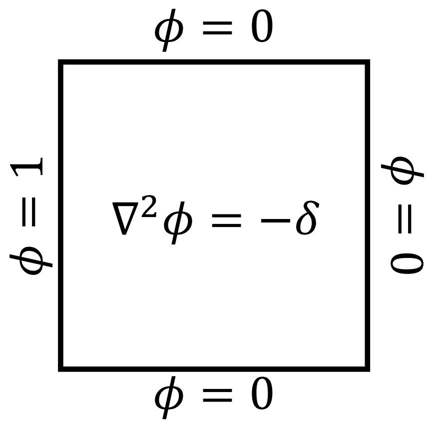



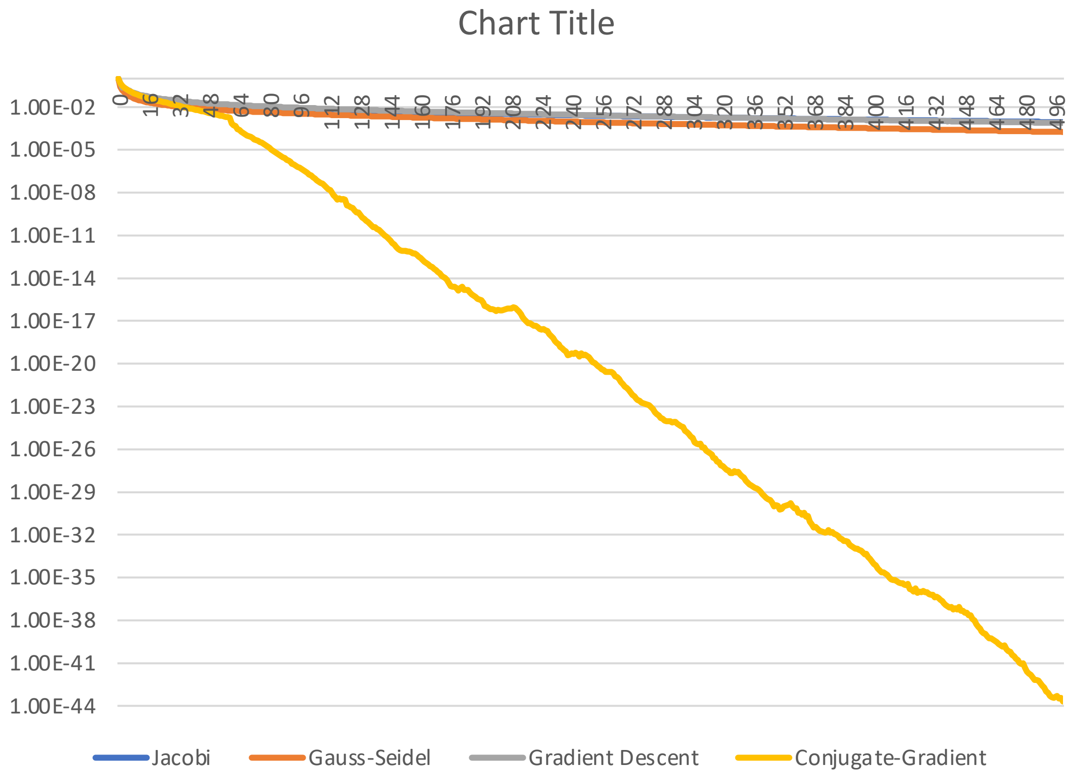



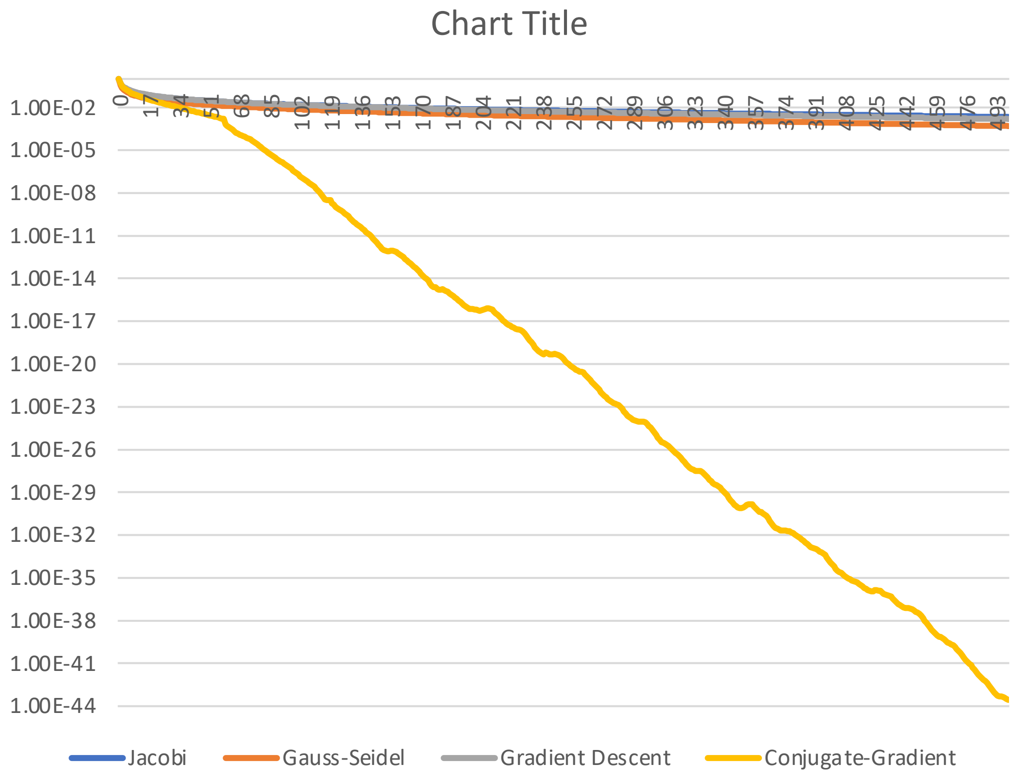



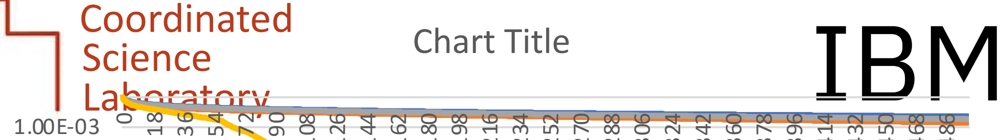
# Verification: Convergence

**60x60x60, h = 0.05**
**27-point equation (HPCG)**



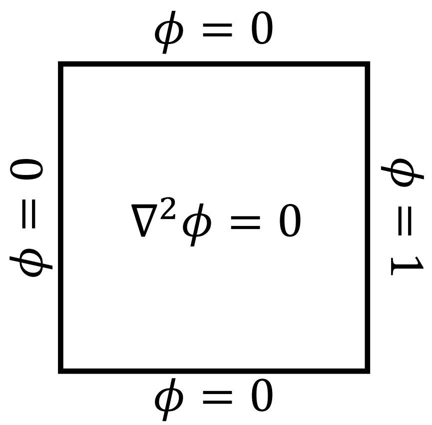



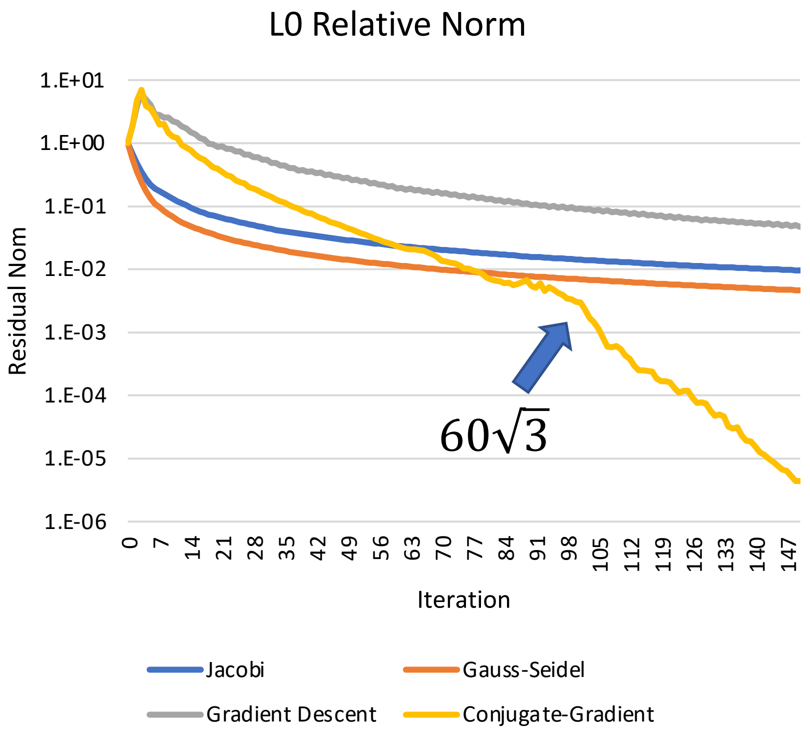



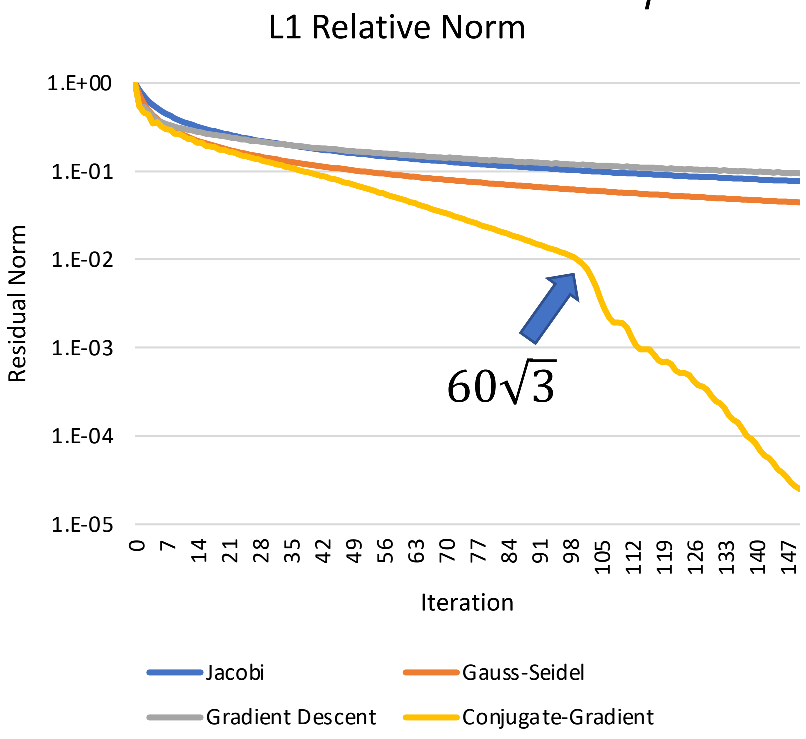
# Verification: Convergence

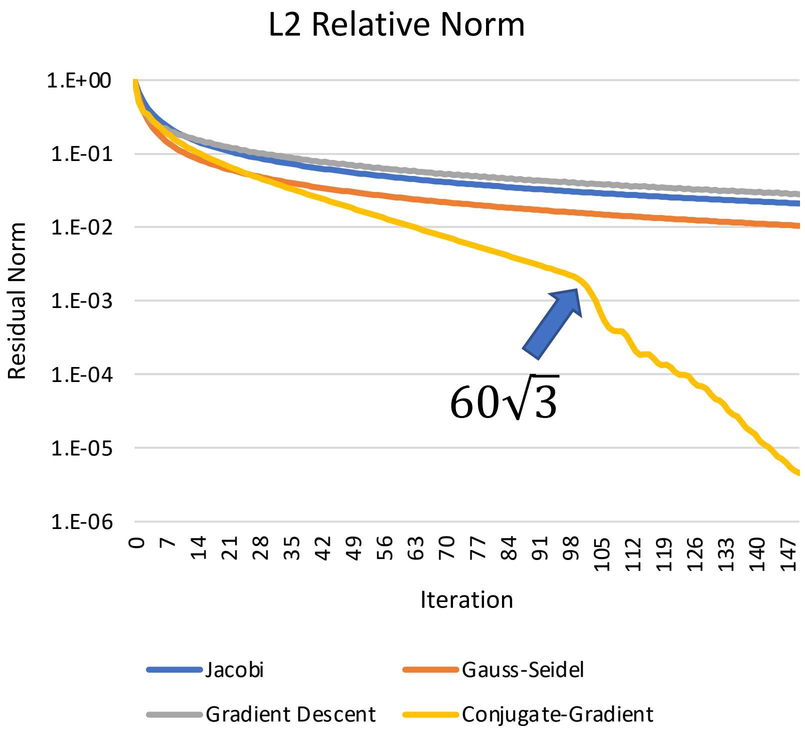
**ZOOMED!**

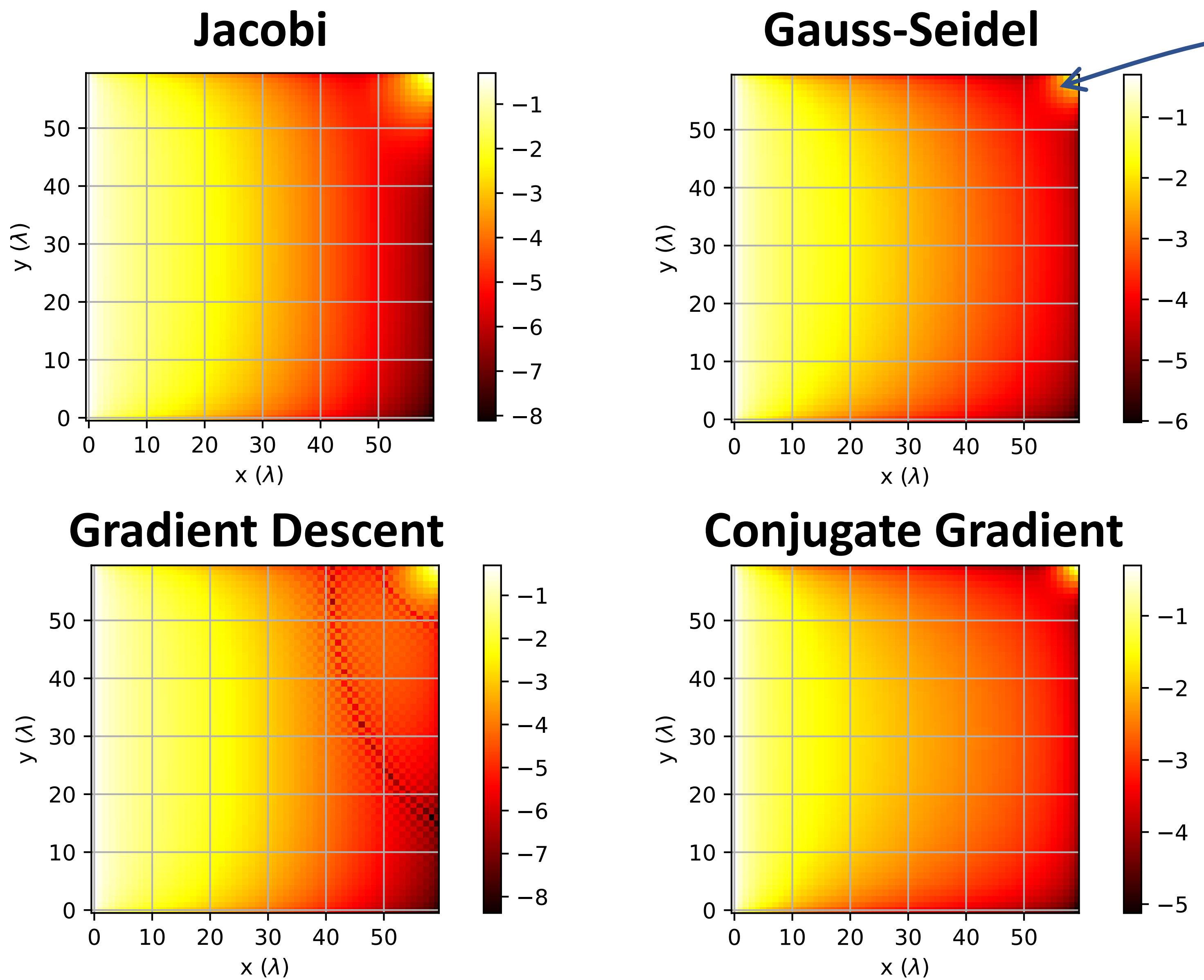


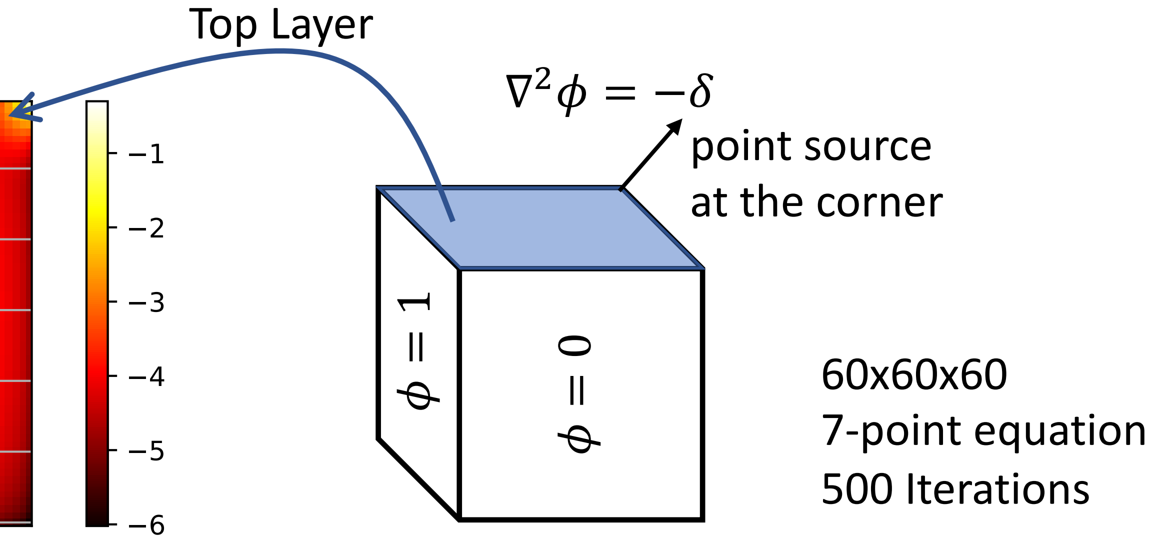

60x60x60
7-point equation

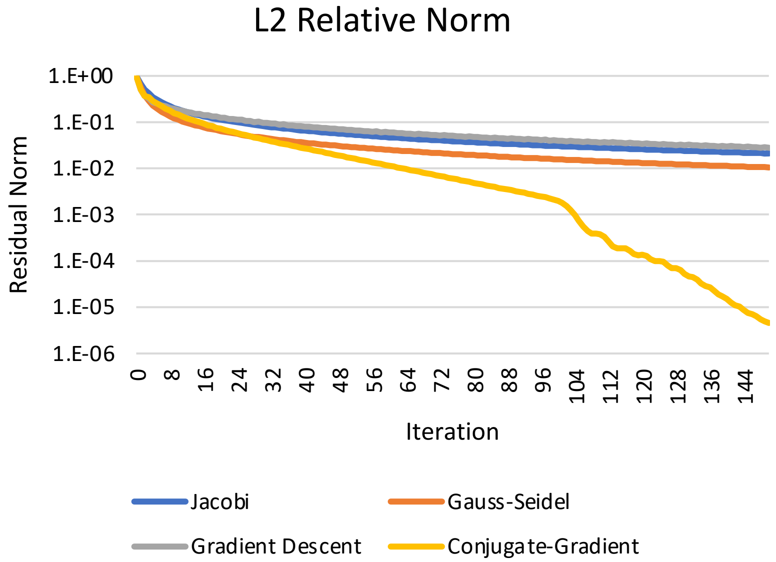





ECE ILLINOIS
Department of Electrical
and Computer Engineering

Coordinated
Science
Laboratory

IBM Research

C3SR
center for
cognitive computing
systems research

# Computational Kernels in Each Iteration

**Other HPCG Operations**

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

**Other HPCG Operations**

```
//GAUSS-SEIDEL BACKWARD SWEEP
for(int batch = numbatch-1; batch > -1; batch--){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL
  for(int m = start+batchsize-1; m > start-1; m--){
    double reduce = b[m];
    for(int n = displ[m+1]-1; n > displ[m]-1; n--)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

**Other HPCG Operations**

```
//SPMV
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM SPMV
  for(int m = start; m < start+batchsize; m++){
    double reduce = 0;
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce += value[n]*buffer[sndex[n]];
    y[m] = reduce;
  }
}
```

# Computational Kernels in Each Iteration

**IDEA: G-S can be fused with SpMV to reuse matrix data from cache**

**Other HPCG Operations**

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

**Other HPCG Operations**

```
//GAUSS-SEIDEL BACKWARD SWEEP
for(int batch = numbatch-1; batch > -1; batch--){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL
  for(int m = start+batchsize-1; m > start-1; m--){
    double reduce = b[m];
    for(int n = displ[m+1]-1; n > displ[m]-1; n--)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

**Other HPCG Operations**

```
//SPMV
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM SPMV
  for(int m = start; m < start+batchsize; m++){
    double reduce = 0;
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce += value[n]*buffer[sndex[n]];
    y[m] = reduce;
  }
}
```

# Computational Kernels in Each Iteration

Other HPCG Operations

```
//GAUSS-SEIDEL FORWARD SWEEP
for(int batch = 0; batch < numbatch; batch++){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL PER BATCH
  for(int m = start; m < start+batchsize; m++){
    double reduce = b[m];
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

Other HPCG Operations

```
//FUSED GAUSS-SEIDEL BACKWARD SWEEP & SPMV
for(int batch = numbatch-1; batch > -1; batch--){
  //READ MAPPING DATA
  int start = mapdispl[batch];
  int mapnz = mapdispl[batch+1]-start;
  //ALLOCATE BUFFER
  double buffer[batchsize+mapnz];
  //LOAD INTER-BATCH DATA
  for(int n = 0; n < mapnz; n++)
    buffer[batchsize+n] = x[map[start+n]];
  //LOAD INTRA-BATCH DATA
  start = batch*batchsize;
  for(int n = 0; n < batchsize; n++)
    buffer[n] = x[start+n];
  //PERFORM GAUSS-SEIDEL
  for(int m = start+batchsize-1; m > start-1; m--){
    double reduce = b[m];
    for(int n = displ[m+1]-1; n > displ[m]-1; n--)
      reduce -= value[n]*buffer[sndex[n]];
    buffer[m-start] += reduce/diag[m];
  }
  //PERFORM SPMV
  for(int mn = spmvdispl[batch]; mn < spmvdispl[batch+1]; mn++){
    int m = spmvmap[mn];
    double reduce = 0;
    for(int n = displ[m]; n < displ[m+1]; n++)
      reduce += value[n]*buffer[sndex[n]];
    y[m] = reduce;
  }
  //WRITE-BACK INTRA-BATCH DATA
  for(int m = 0; m < batchsize; m++)
    x[start+m] = buffer[m];
}
```

Other HPCG Operations

## To-Do List:

- Validate correctness by solving a 3-D Laplace equation
- Investigate convergence vs. speedup trade-off
- Incorporate kernels into HPCG and validate again
- Scale on Summit ???