



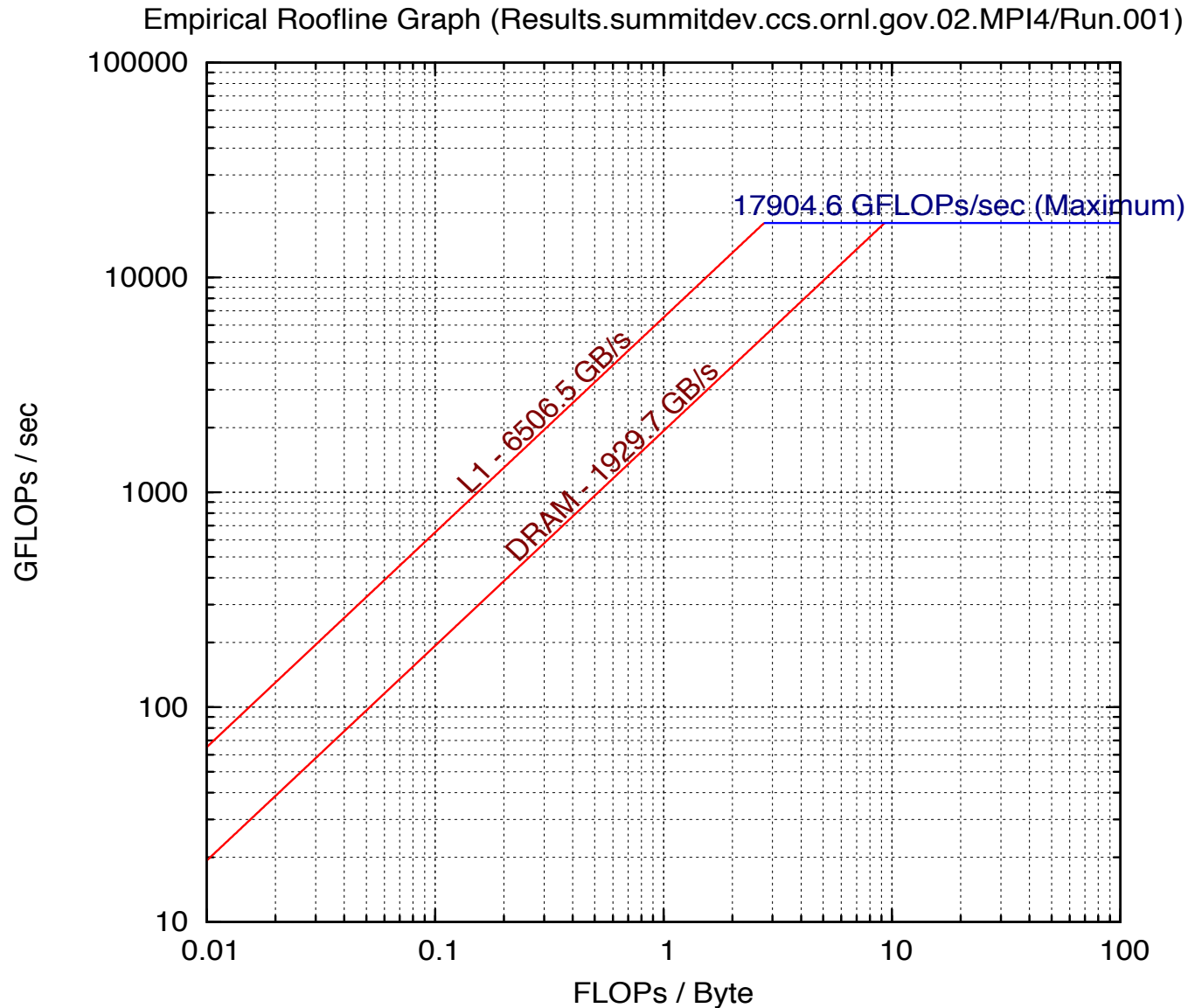
BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY



Using Empirical Roofline Toolkit and Nvidia nvprof

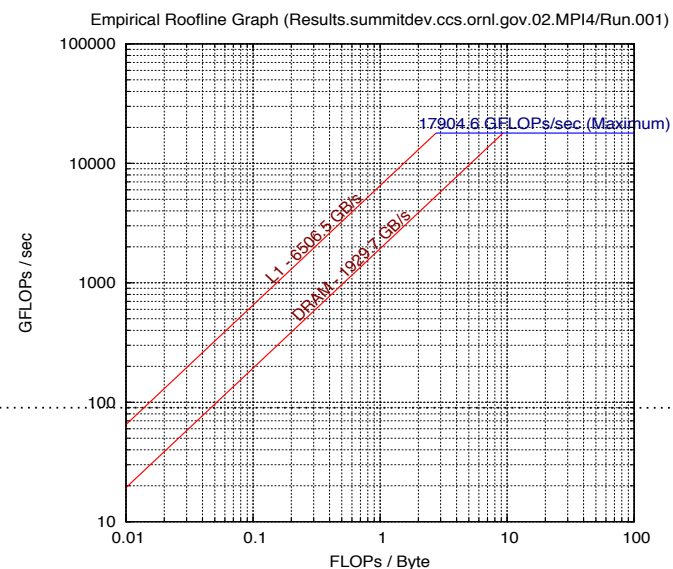
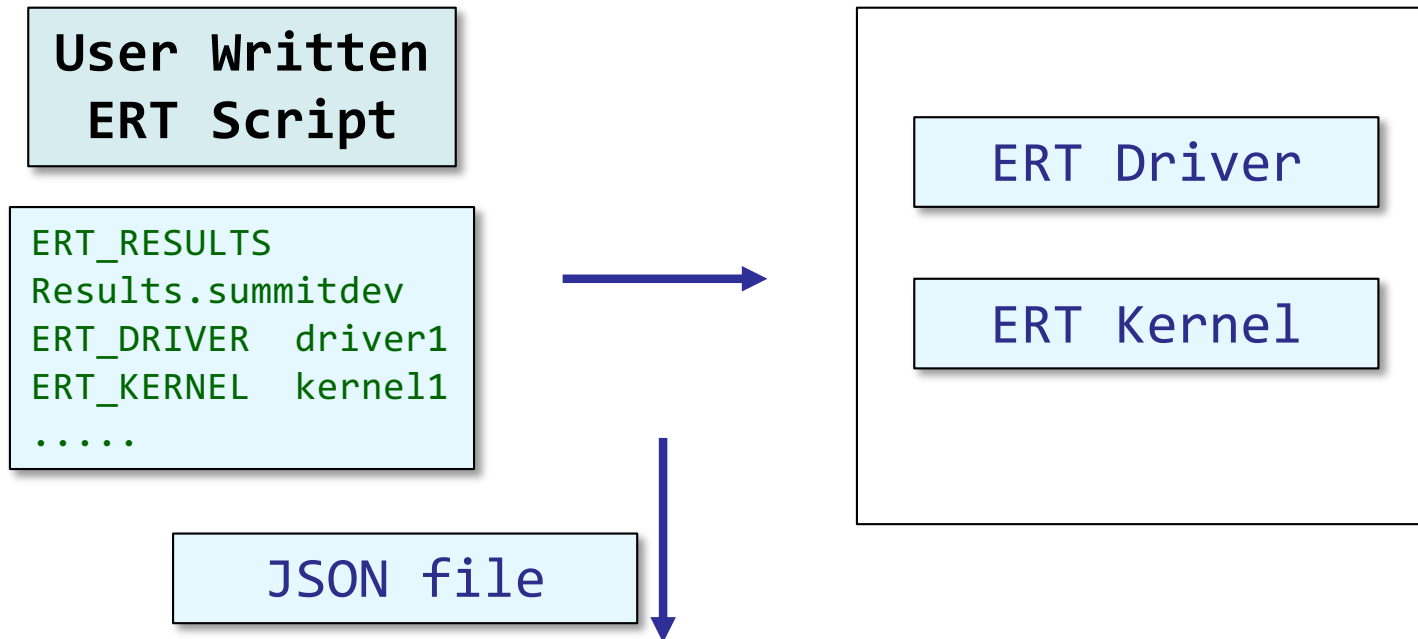
Protonu Basu, Samuel Williams, Leonid Oliker
Lawrence Berkeley National Laboratory

ERT Results from a SummitDev Node



<https://bitbucket.org/berkeleylab/cs-roofline-toolkit>

Components of the ERT



Driver

Driver

```
int main () {  
    #pragma omp parallel private(id)  
    {  
        uint64_t n, t;  
        initialize(&A[nid]);  
        for (n = 16; n < SIZE; n *= 1.1) {  
            for (t = 1; t < TRIALS; t *= 2) {  
                // start timer here  
                Kernel(n, t, &A[nid]);  
                // stop timer here  
                #pragma omp barrier  
                #pragma omp master  
                {  
                    MPI_Barrier(MPI_COMM_WORLD);  
                }  
            }  
        }  
    }  
}
```

Init

Compute

Sync

Kernel

Bandwidth

```
void Kernel (uint64_t size, uint64_t trials,
double * __restrict__ A) {
    double alpha = 0.5;
    uint64_t i, j;
    for (j = 0; j < trials; ++j ) {
        for (i = 0; i < nsize; ++i) {
            A[i] = A[i] + alpha;
        }
        alpha = alpha * 0.5;
    }
}
```

Kernel

GFlops

```
void Kernel (uint64_t size, uint64_t trials,
double * __restrict__ A) {
    double alpha = 0.5;
    uint64_t i, j;
    for (j = 0; j < trials; ++j ) {
        for (i = 0; i < nsize; ++i) {
            double bette = 0.8;
            #if FLOPPERITER == 2
            beta = beta * A[i] + alpha;
            #elif FLOPPERITER == 4
            beta = beta * A[i] + alpha;
            beta = beta * A[i] + alpha;
            #elif FLOPPERITER == 8
            ...
            #endif
            A[i] = beta;
        }
        alpha = alpha * 0.5;
    }
}
```

Configuration Script for SummitDev

```
ERT_RESULTS Results.summitdev
ERT_DRIVER  driver1
ERT_KERNEL  kernel1

ERT_GPU      True
ERT_GPU_CFLAGS -x cu

ERT_MPI True
ERT_FLOPS    1,2,4,8,16,32,64,128,256
ERT_ALIGN    32

ERT_CC      nvcc
ERT_CFLAGS  -O3 -gencode code=sm_60 ...
ERT_LD      mpicc
ERT_LDFLAGS ...

ERT_MPI_PROC 4

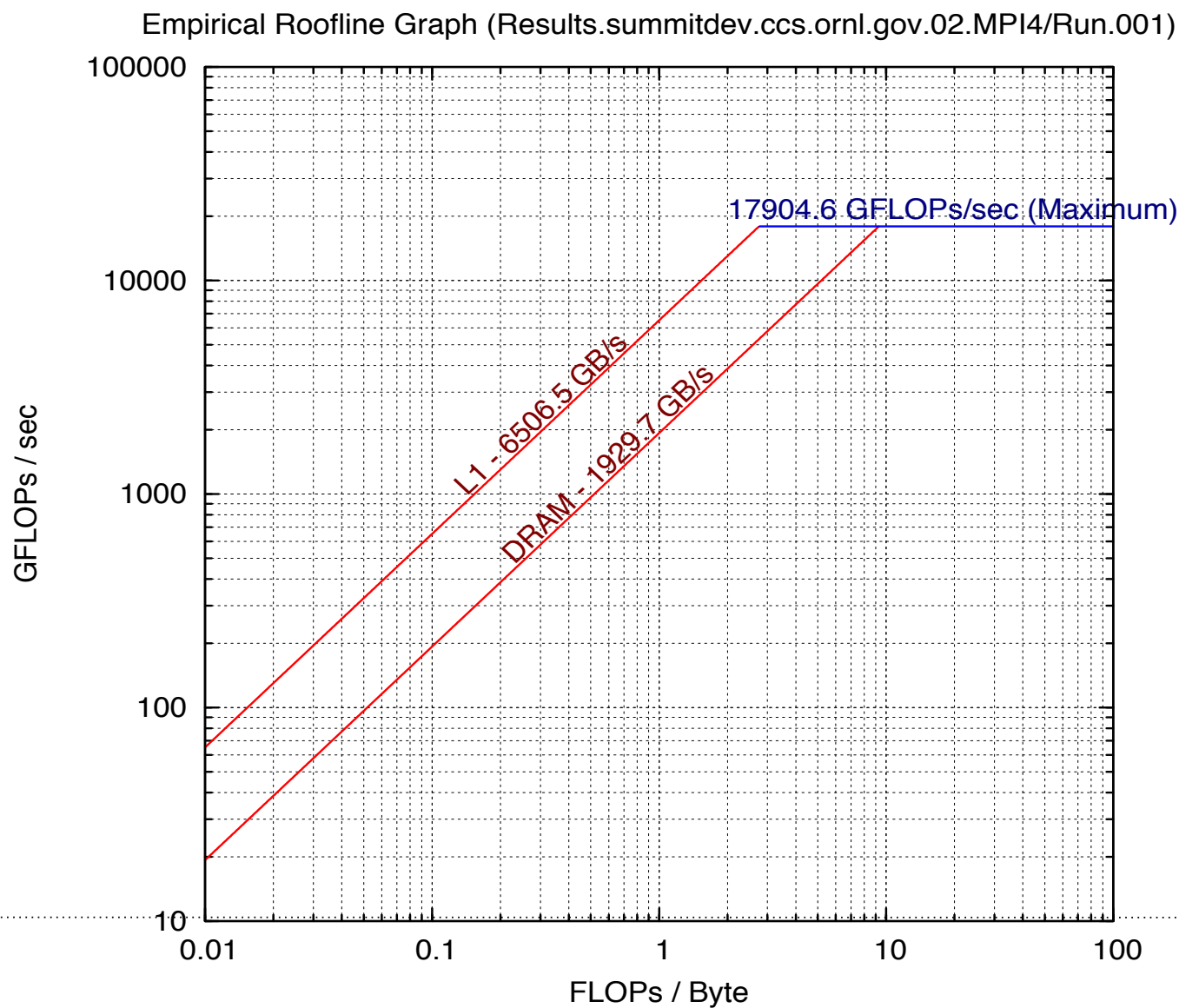
ERT_RUN      mpirun -np ERT_MPI_PROCS -map-by
ppr:4:node --bind-to none -gpu -display-map
ERT_CODE
```

```
ERT_BLOCKS_THREADS 28672
ERT_GPU_BLOCKS      28,56,112,224,448
ERT_GPU_THREADS      64,128,256,512,1024

ERT_NUM_EXPERIMENTS 1
ERT_MEMORY_MAX 1073741824
ERT_WORKING_SET_MIN 128
ERT_TRIALS_MIN 1

ERT_GNUPLOT gnuplot
```

ERT Results from a SummitDev Node



AI of using Nvidia nvprof

Nvidia nvprof enables the collection of a timeline of CUDA-related activities on both CPU and GPU, including kernel execution, memory transfers, memory set and CUDA API calls

AI of using Nvidia nvprof

To plot roofline (AI) data for a kernel, we need:

Number of floating-point operations executed

Data-volume to and from DRAM or cache

The runtime in seconds

Runtime of Kernel

```
command: nvprof --print-gpu-trace ./build/bin/hpgmg-fv 6 8
```

output:

Time	Calls	Avg	Min	Max
2.52256s	1764	1.4300ms	1.4099ms	1.4479ms

Name

void smooth_kernel<int=7,int=16,int=4,int=16>
(level_type, int, int, double, double, int, double*, double*)

Nvidia nvprof is used in the gpu-trace mode

Using nvprof to Collect Metrics

Use the nvprof metric summary mode to collect:

Flops

DRAM read/write throughput

DRAM read/write transactions

```
command: nvprof --kernels "smooth_kernel"  
--metrics flop_count_dp  
--metrics dram_read_throughput  
--metrics dram_write_throughput  
--metrics dram_read_transactions  
--metrics dram_write_transactions  
./build/bin/hpgmg-fv 6 8
```

Using nvprof to Collect Metrics

```
command: nvprof --kernels "smooth_kernel"  
--metrics flop_count_dp  
--metrics dram_read_throughput  
--metrics dram_write_throughput  
--metrics dram_read_transactions  
--metrics dram_write_transactions  
./build/bin/hpgmg-fv 6 8
```

Kernel: void smooth_kernel<int=7, int=32, int=4, int=16>(level_type, int, int, double, double, int, double*, double*)

Invocations	Metric Name	Min	Max	Avg
1764	flop_count_dp	240648192	240648192	240648192
1764	dram_read_throughput	299.98 GB/s	307.48GB/s	303.72 GB/s
1764	dram_write_throughput	40.102GB/s	41.099GB/s	40.578GB/s
1764	dram_read_transactions	4537918	4599890	4567973
1764	dram_write_transactions	606387	611691	610299

Computing Arithmetic Intensity

Method 1

$$FP / (DR + DW) * (\text{size of transaction} = 32 \text{ Bytes})$$

Method 2

$$FP / (TR + TW) * \text{time taken by kernel}$$

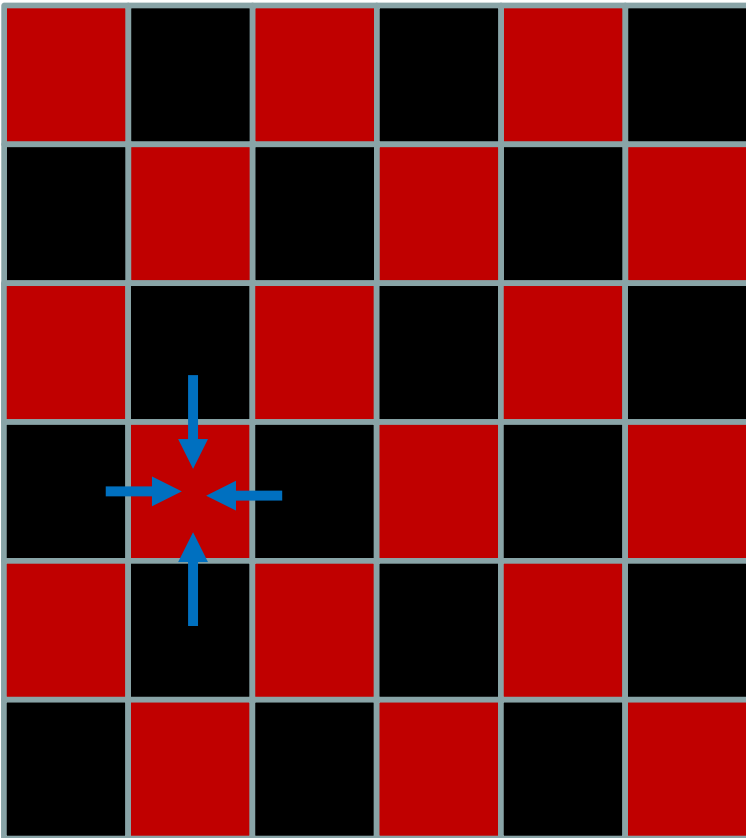
FP = double precision ops

DR/DW= dram read/write transactions

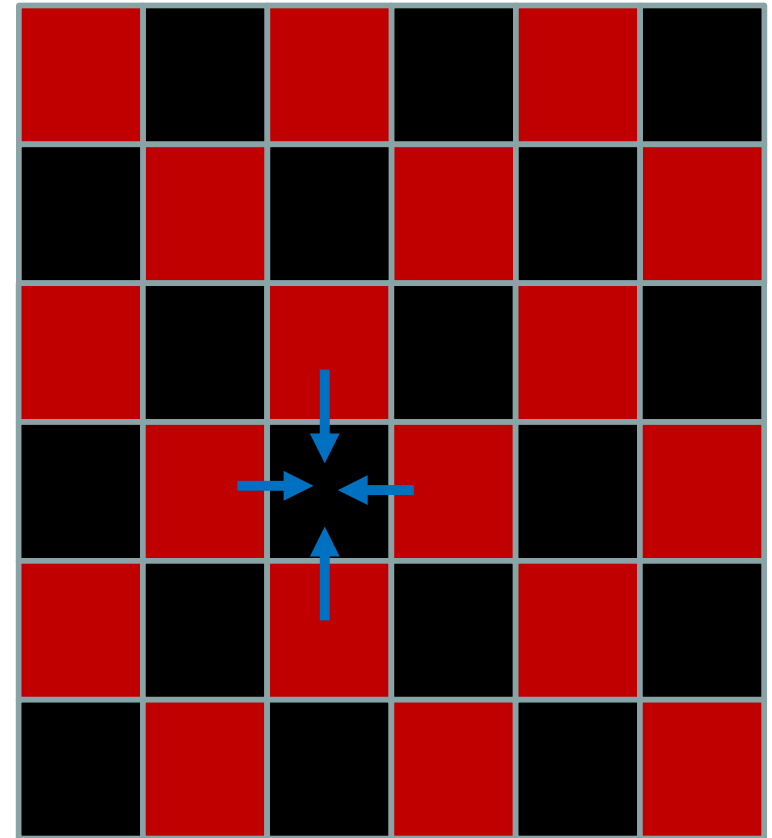
TR/TW= dram read/write throughput

Gauss-Seidel Red-Black (GSRB)

Red Sweep: update red points



Black Sweep: update black points



Two Flavors of GSRB (GSRB_BRANCH)

```
template<int LOG_DIM_I, int BLOCK_I, int BLOCK_J, int BLOCK_K>
__global__ void smooth_kernel(...) {

    int i = ilo + threadIdx.x;
    int j = jlo + threadIdx.y;

    for (k = klow; k < klow + kdim; k++) {
        int ijk = i + j * jStride + k * kStride;
        if ((i ^ j ^ k ^ color ^ 1) & 1) {
            double Ax = apply_op_ijk(); // 7-point VC stencil
            double lambda = Dinv_ijk();
            xo[ijk] = X(ijk) + lambda * (rhs[ijk] - Ax);
        }
    }
}
```

The if-condition blocks unnecessary computation

Two Flavors of GSRB (GSRB_FP)

```
template<int LOG_DIM_I, int BLOCK_I, int BLOCK_J, int BLOCK_K>
__global__ void smooth_kernel(...) {

    int i = ilo + threadIdx.x;
    int j = jlo + threadIdx.y;

    for (k = klow; k < klow + kdim; k++) {
        int ijk = i + j * jStride + k * kStride;
        double* RedBlack = level.RedBlack_FP
            + ghosts * (1 + jStride)
            + ((k ^ color000) & 1) * kStride;
        double Ax = apply_op_ijk(); // 7-point VC stencil
        double lambda = Dinv_ijk();
        xo[ijk] = X(ijk) + RedBlack[ij] * lambda * (rhs[ijk] - Ax);
    }
}
```

RedBlack mask is used to block unnecessary writes

Results from GSRB on SummitDev GPUs

Thread Block (I=32, J=4)

	flops executed	Read Transactions	Write Transactions	Data Movement (Bytes)
GSRB_FP	240648192	4566618	610123	165655712
GSRB_BRANCH	120061952	9306276	613283	317425888

ERT BANDWIDTH : 497 GB/s

GSRB_FP READ+WRITE THROUGHPUT: 371 GB/s

Results from GSRB on SummitDev GPUs

Thread Block (I=32, J=4)

	flops executed	Read Transactions	Write Transactions	Data Movement (Bytes)
GSRB_FP	240648192	4566618	610123	165655712
GSRB_BRANCH	120061952	9306276	613283	317425888

	flops (requisite)	Ratio	Data Movement (Bytes) Requisite	Ratio
GSRB_FP	119537664	2.013157895	128798208	1.286164727
GSRB_BRANCH	119537664	1.004385965	110398464	2.875274497

Requisite flops computed for GSRB_branch