Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

Performance Analysis of Common Loop Optimizations

Brian J Gravelle
Mentor: Dave Nystrom
HPC-ENV

Aug. 2021

LA-UR-21-28269

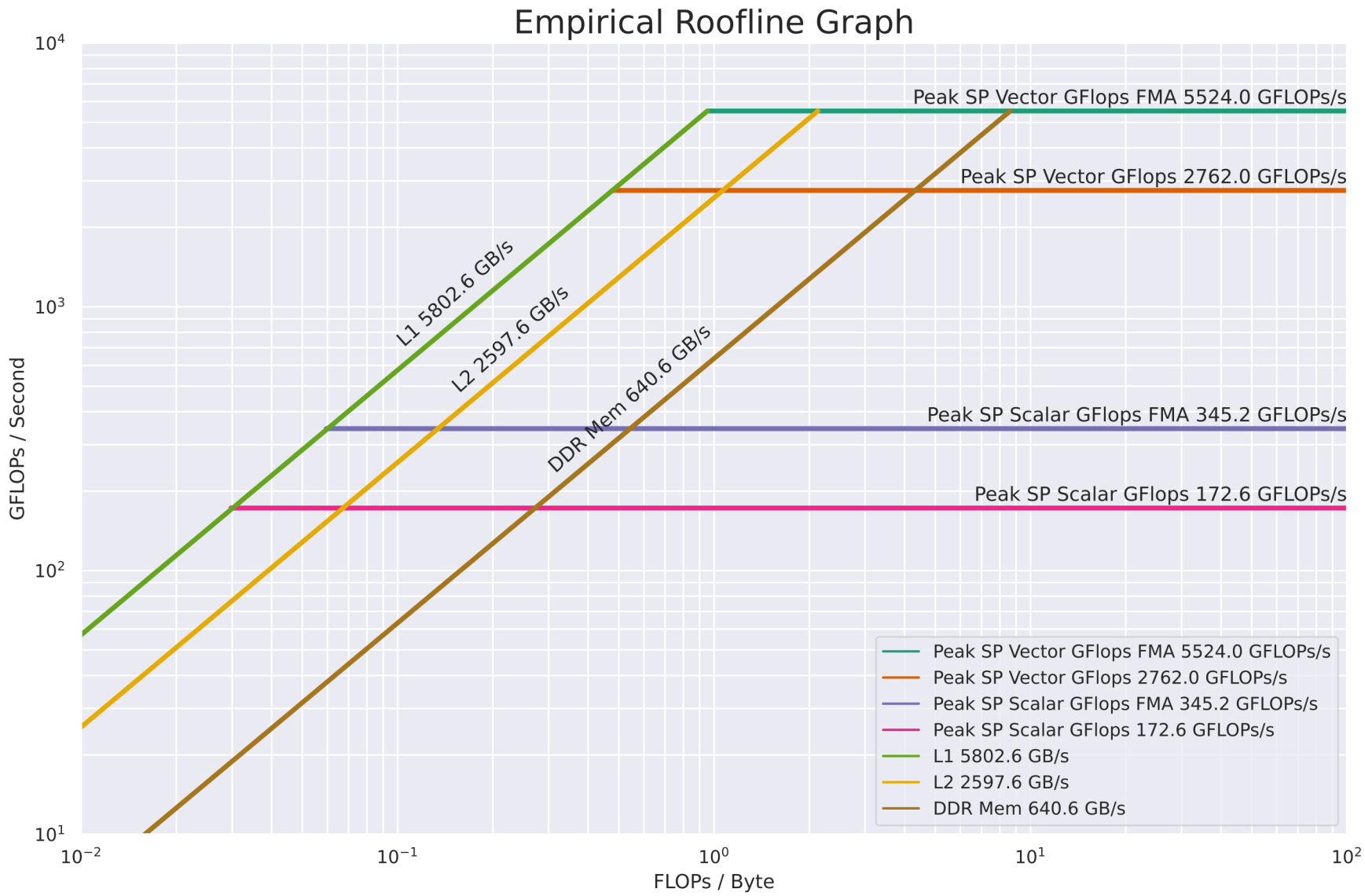
Why performance analysis?

How does an application make use of the Supercomputer?

```
!$OMP DO PRIVATE(right_flux,left_flux,top_flux,bottom_flux,total_flux,min_cell_volume, &
!$OMP             energy_change,recip_volume,volume_change_s)
DO k=y_min,y_max
  DO j=x_min,x_max

    left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,k+1)                    &
      +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
    right_flux= (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                    &
      +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
    bottom_flux=(yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                    &
      +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
    top_flux=   (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                    &
      +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
    total_flux=right_flux-left_flux+top_flux-bottom_flux

    volume_change_s=volume(j,k)/(volume(j,k)+total_flux)

    min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
      ,volume(j,k)+right_flux-left_flux                      &
      ,volume(j,k)+top_flux-bottom_flux)

    recip_volume=1.0/volume(j,k)

    energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*total_flux*recip_volume

    energy1(j,k)=energy0(j,k)-energy_change

    density1(j,k)=density0(j,k)*volume_change_s

  ENDDO
ENDDO
!$OMP END DO
```

And how can we make it run faster?

Roofline Analysis



Roofline Analysis

- Arithmetic Intensity: Flops/Byte
  - Measure Floating point operations
  - Bytes of data moved

- Computation Rate: Flops / Second
  - Measure Floating point operations
  - Time

Hardware Performance Monitors

- HPMs are used for performance analysis
  - Hardware Counters
  - Bridge the gap between application and hardware
  - Cover numerous features of the hardware
  - Count events that occur while an application runs

Our work

- Merge hardware counter analysis and Roofline analysis
  - Clear visualization
  - Detailed information

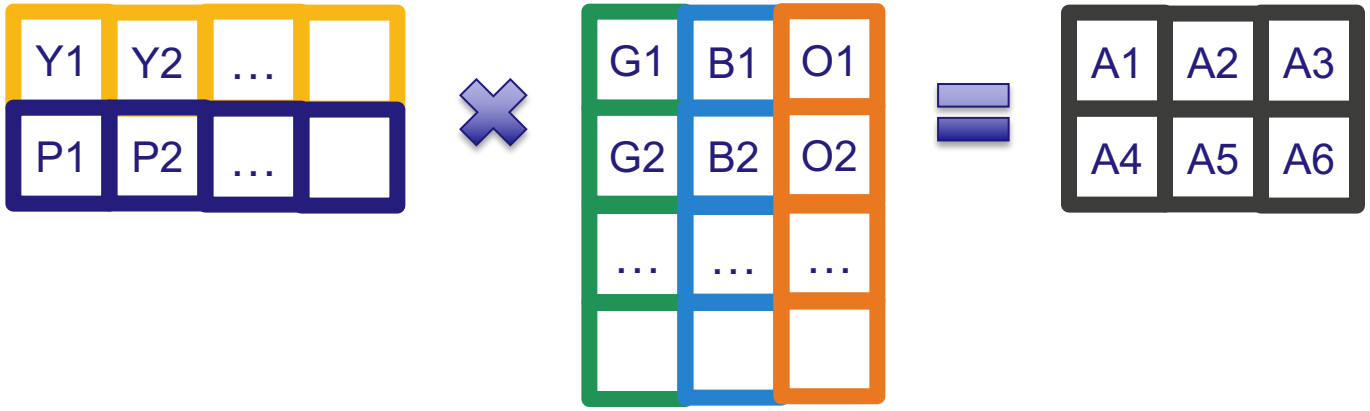
- Two parts
  - Counters to plot application points
  - Counters to understand application points

Methodology

- Systems
  - A64FX – Fujitsu's ARM processor with 512 bit SVE
    - 48 Cores, 32 GB of HBM
  - ARM HPC compiler, Cray compiler

- Caliper and PAPI
  - Hardware counter collection

A Basic Example: Matrix Multiplication



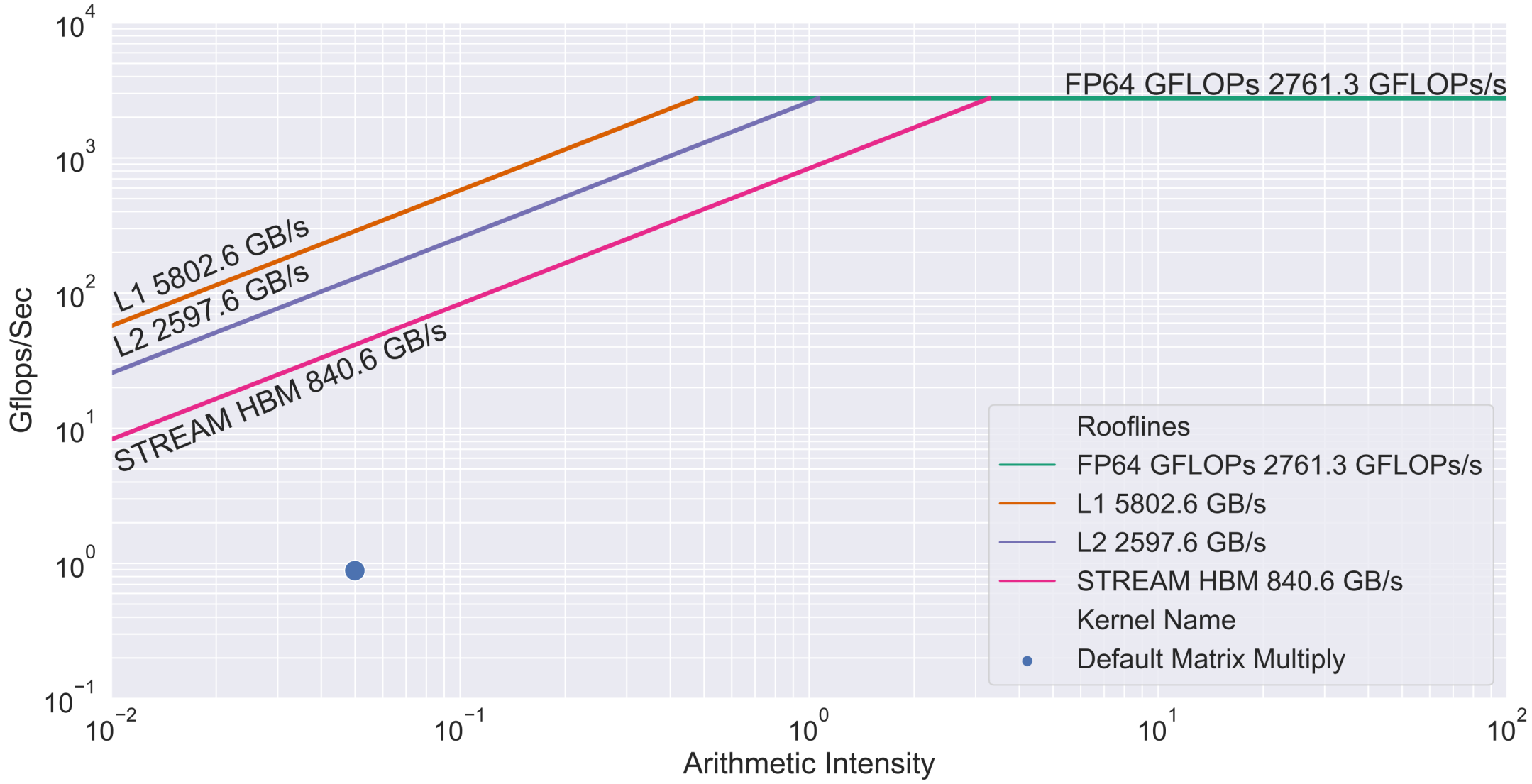
A1 = Y1*G1 + Y2*G2 + …

Memory Layout:

A Basic Example: Matrix Multiplication

```
#pragma omp parallel for
for (int i = 0; i < rows_a; i++ )
    for (int j = 0; j < cols_b; j++)
        for (int k = 0; k < cols_a; k++)
            mat_c[i][j] = mat_c[i][j] + mat_a[i][k] * mat_b[k][j];
```

A Basic Example: Matrix Multiplication



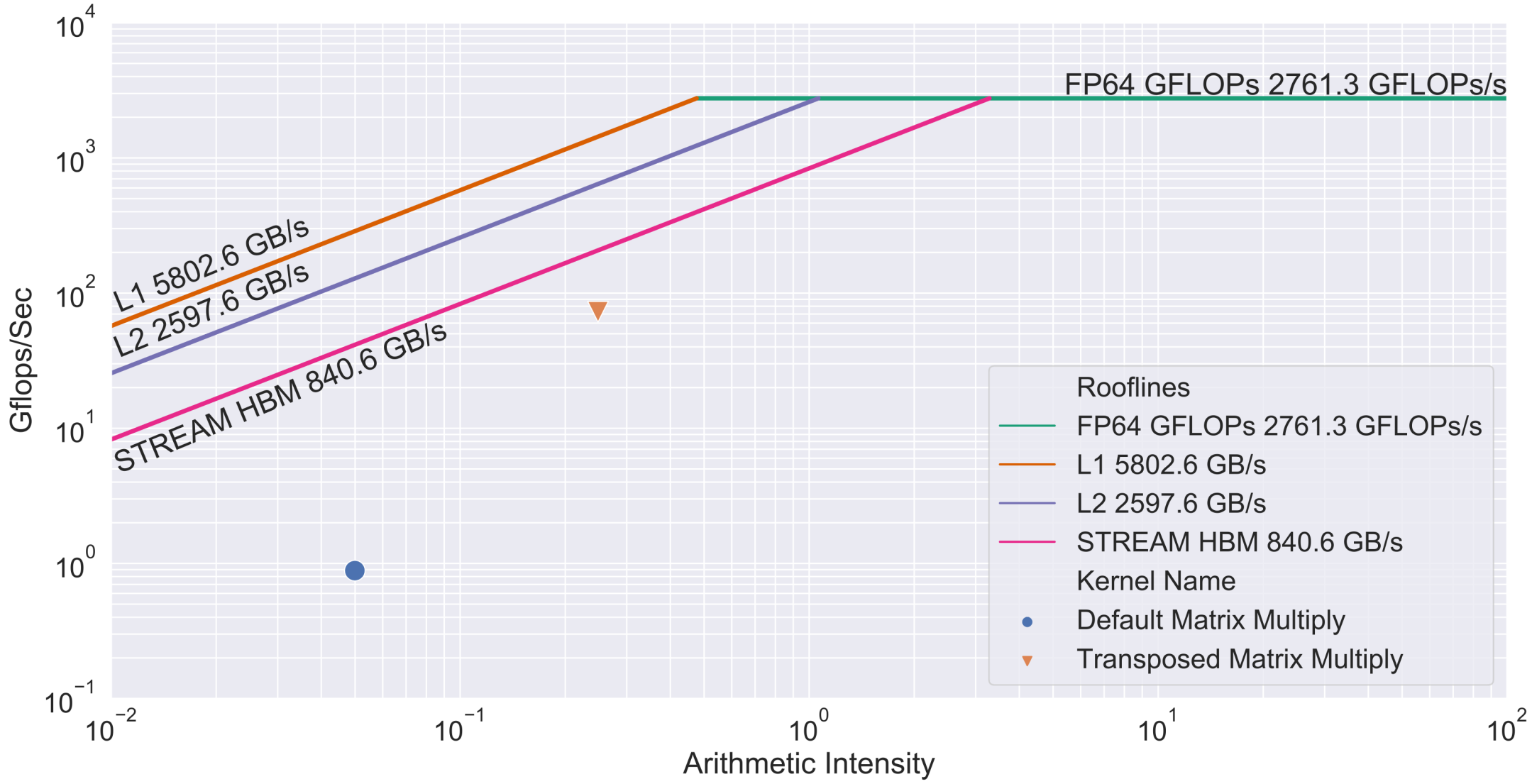
A Basic Example: Matrix Multiplication

	Default
Time (s)	1242.83
Arithmetic Intensity	0.05
Mem bytes / LS bytes	0.14
L2 bytes / LS bytes	7.38
FLOPs per FP Instruction	2.00
EU Stall fraction	0.03
Load Stall fraction	0.97

Optimization: Transposed

```
#pragma omp parallel for
for (int i = 0; i < rows_a; i++ )
    for (int j = 0; j < cols_b; j++)
        for (int k = 0; k < cols_a; k++)
            mat_c[i][j] = mat_c[i][j] + mat_a[i][k] * mat_b[j][k];
```

Optimization: Transposed



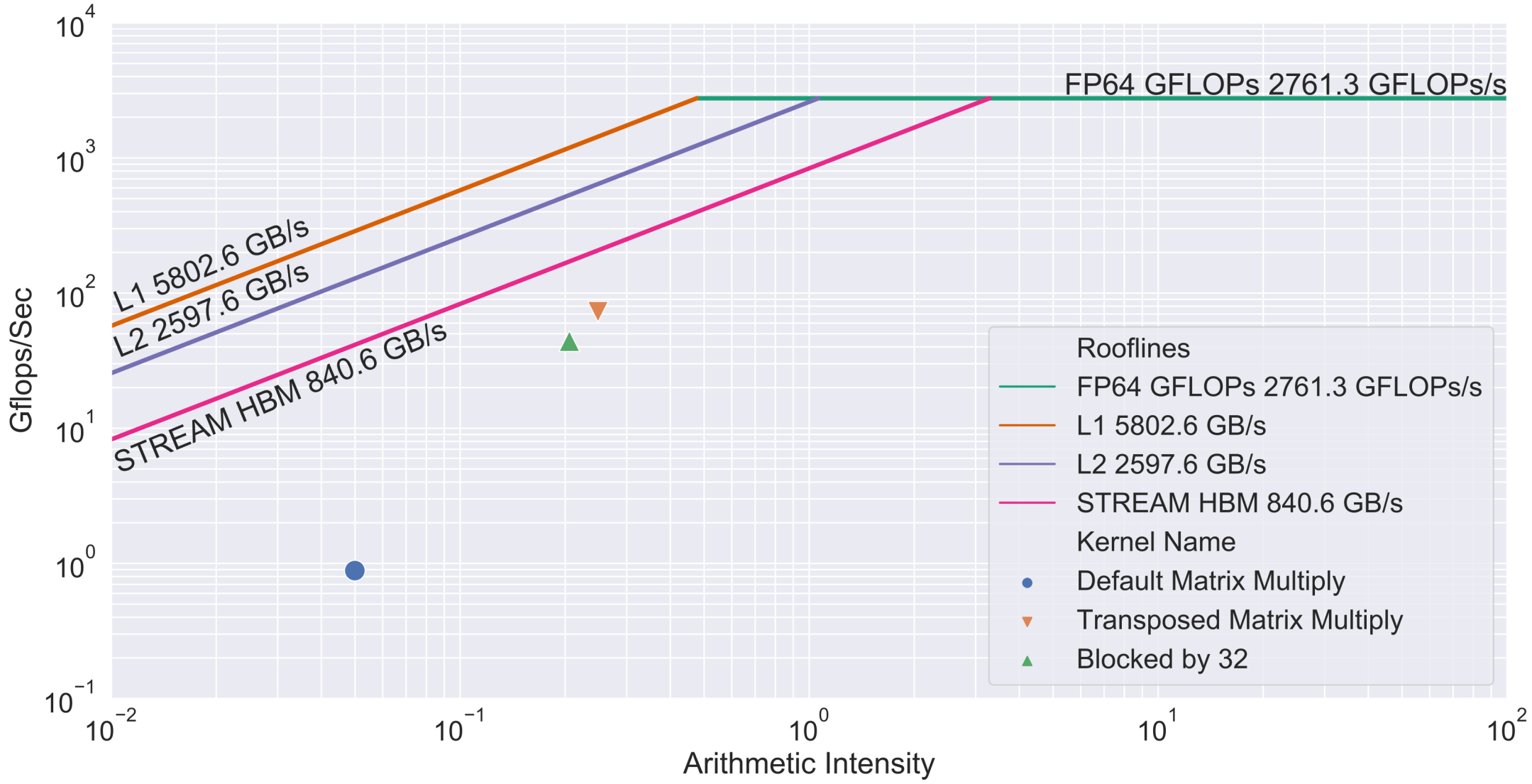
Optimization: Transposed

	Default	Transposed
Time (s)	1242.83	15.11
Arithmetic Intensity	0.05	0.25
Mem bytes / LS bytes	0.14	0.86
L2 bytes / LS bytes	7.38	3.13
FLOPs per FP Instruction	2.00	2.00
EU Stall fraction	0.03	0.91
Load Stall fraction	0.97	0.08

Optimization: Blocked

```
#pragma omp parallel for
for (int i = 0; i < rows_a; i++ )

  for (int jjj = 0; jjj < cols_b; jjj = jjj + BLOCK_ROWS)
    for (int j = jjj; j < min(cols_b, jjj + BLOCK_ROWS); j++)

      for (int kkk = 0; kkk < cols_a; kkk = kkk + BLOCK_COLS)
        for (int k = kkk; k < min(cols_a,kkk + BLOCK_COLS); k++)

          mat_c[i][j] = mat_c[i][j] + mat_a[i][k] * mat_b[j][k];
```

Optimization: Blocked



Optimization: Blocked

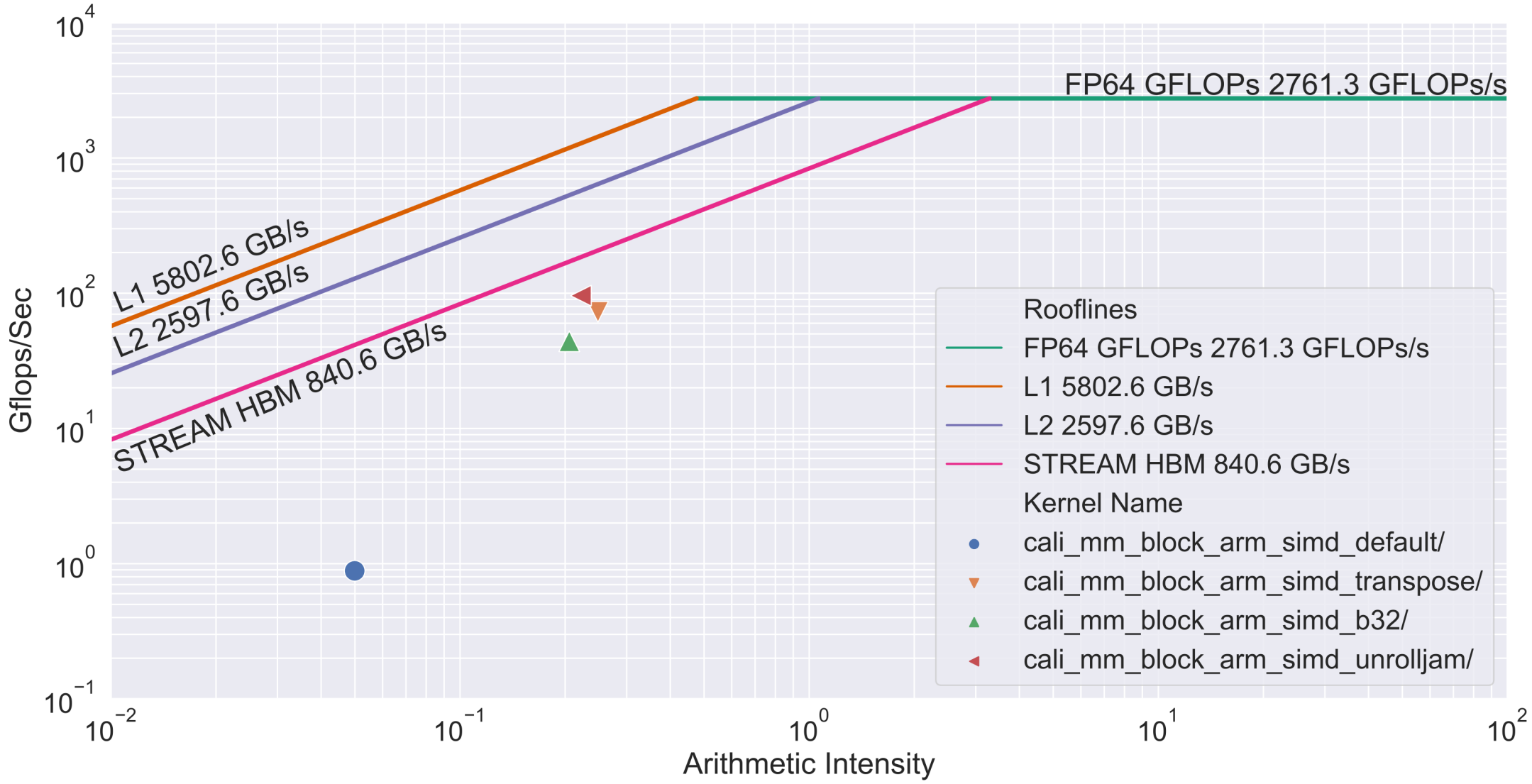
	Default	Transposed	Blocked
Time (s)	1242.83	15.11	26.52
Arithmetic Intensity	0.05	0.25	0.21
Mem bytes / LS bytes	0.14	0.86	0.54
L2 bytes / LS bytes	7.38	3.13	2.45
FLOPs per FP Instruction	2.00	2.00	2.00
EU Stall fraction	0.03	0.91	0.78
Load Stall fraction	0.97	0.08	0.22

Optimization: Unrolled

```
#pragma omp parallel for
for (int i = 0; i < rows_a; i++ ) {
    int j;
    for (j = 0; j < cols_b-8; j+=8) {

        #pragma omp simd
        for (int k = 0; k < cols_a; k++){
          mat_c[i][j]   = mat_c[i][j]   + mat_a[i][k] * mat_b[j][k];
          mat_c[i][j+1] = mat_c[i][j+1] + mat_a[i][k] * mat_b[j+1][k];
          mat_c[i][j+2] = mat_c[i][j+2] + mat_a[i][k] * mat_b[j+2][k];
          mat_c[i][j+3] = mat_c[i][j+3] + mat_a[i][k] * mat_b[j+3][k];
          mat_c[i][j+4] = mat_c[i][j+4] + mat_a[i][k] * mat_b[j+4][k];
          mat_c[i][j+5] = mat_c[i][j+5] + mat_a[i][k] * mat_b[j+5][k];
          mat_c[i][j+6] = mat_c[i][j+6] + mat_a[i][k] * mat_b[j+6][k];
          mat_c[i][j+7] = mat_c[i][j+7] + mat_a[i][k] * mat_b[j+7][k];
        }

    }
    for (j; j < cols_b; j++) {
        #pragma omp simd
        for (int k = 0; k < cols_a; k++)
          mat_c[i][j] = mat_c[i][j] + mat_a[i][k] * mat_b[j][k];
    }
} // i
```

HIGH PERFORMANCE COMPUTING

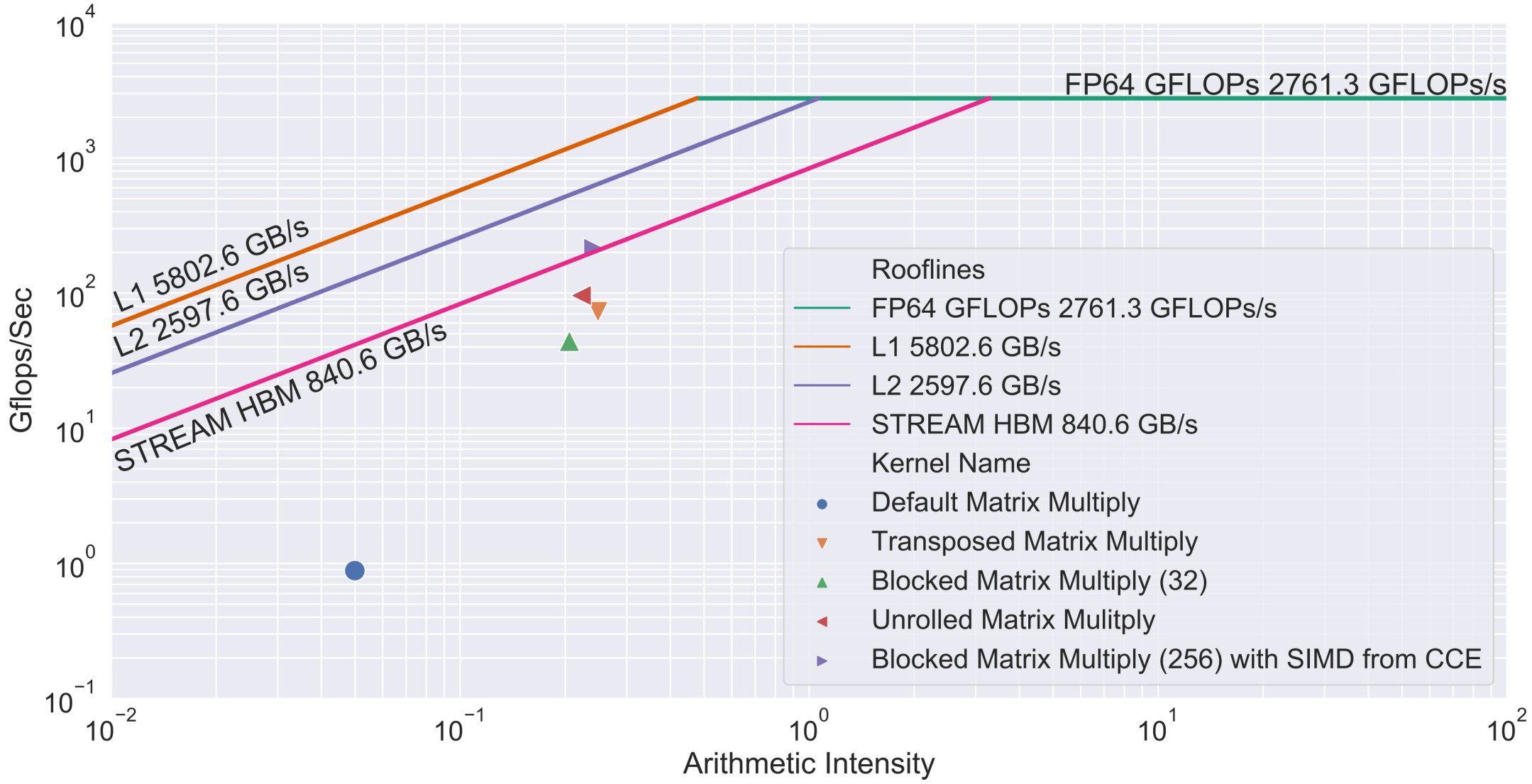
Optimization: Unrolled



Optimization: Unrolled

	Default	Transposed	Blocked	Unrolled
Time (s)	1242.83	15.11	26.52	11.47
Arithmetic Intensity	0.05	0.25	0.21	0.22
Mem bytes / LS bytes	0.14	0.86	0.54	0.13
L2 bytes / LS bytes	7.38	3.13	2.45	2.64
FLOPs per FP Instruction	2.00	2.00	2.00	2.00
EU Stall fraction	0.03	0.91	0.78	0.40
Load Stall fraction	0.97	0.08	0.22	1.10

Optimization: SIMD



Optimization: Unrolled

	Default	Transposed	Blocked	Unrolled	SIMD (CCE)
Time (s)	1242.83	15.11	26.52	11.47	5.17
Arithmetic Intensity	0.05	0.25	0.21	0.22	0.22 (est.)
Mem bytes / LS bytes	0.14	0.86	0.54	0.13	NA
L2 bytes / LS bytes	7.38	3.13	2.45	2.64	NA
FLOPs per FP Instruction	2.00	2.00	2.00	2.00	14.9 (est.)
EU Stall fraction	0.03	0.91	0.78	0.40	NA
Load Stall fraction	0.97	0.08	0.22	1.10	NA

Conclusions

- HPM (counters) can enable roofline analysis
  - Plot application points
  - Reveal how the kernels use the hardware

- Future (and concurrent) work
  - Apply the method to Intel processors
  - … to Cloverleaf
  - … VPIC and Pagosa

Questions?

Over 70 years at the forefront of supercomputing

Contact: Brian J Gravelle, gravelle@lanl.gov

Over 70 years at the forefront of supercomputing