On the Mitigation of Cache Hostile Memory Access Patterns on Many-core CPU Architectures

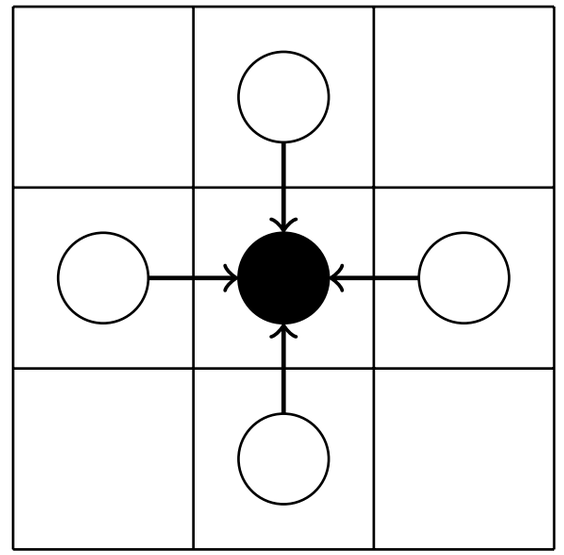
Tom Deakin, Simon McIntosh-Smith: University of Bristol

Acknowledgements: John Pennycook, Andy Mallinson: Intel Corporation
Thanks for Cray Inc. for access to the Cray XC40 Supercomputer “Swan”
The University of Bristol is an Intel Parallel Computer Center

Stencil patterns

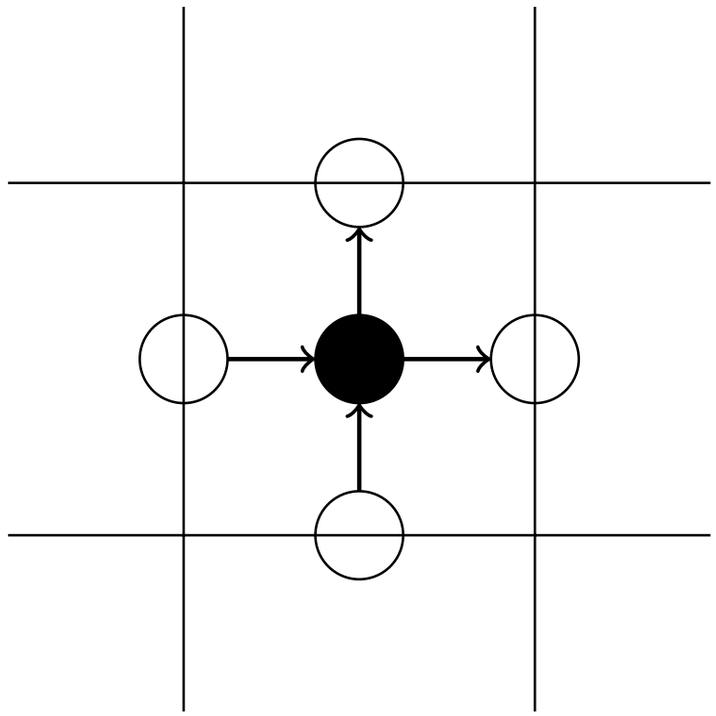
5-point stencil

- Reads data from (i±1, j±1)
- Used in lots of finite difference codes (e.g. Lattice Boltzmann)

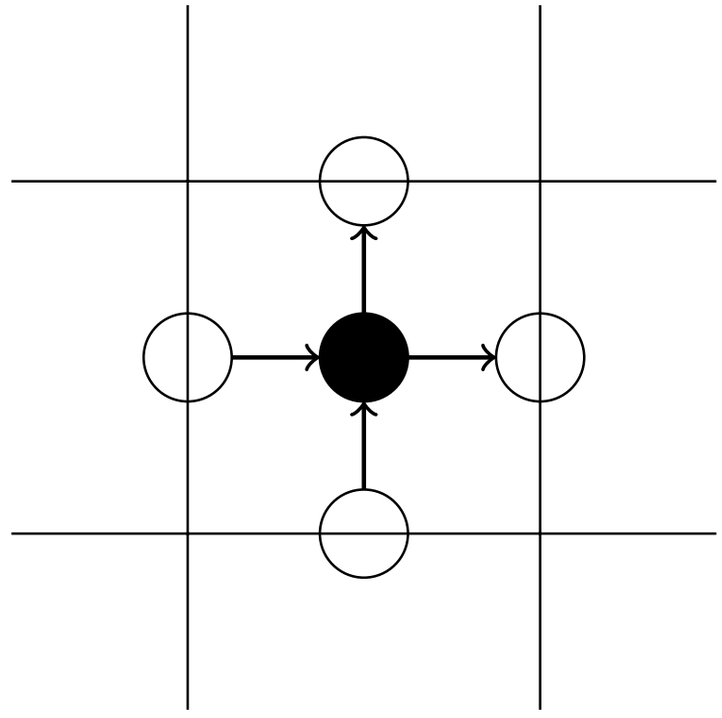


<u>Upwinded</u> 5-point stencil

- Reads data from (i-1,j-1)
- Writes data to (i+1, j+1)
- Used in for e.g. LU factorisation, deterministic transport
- Results in sweep across mesh



Computational kernel



1. Calculate cell centred value**s**
2. Calculate outgoing face values
3. Reduce cell centred values within each cell

- All operations have low computational intensity
- Sit in the memory bandwidth bound section in the Roofline model

University of BRISTOL

Memory access patterns

- Multiple cell centred values in each cell
  - Store as inner-most dimension - stride 1 in large mesh array
    - Compiler auto-vectorisation
    - A few FMAs; 1 flop/double
    - Stream through large mesh, no reuse
- Edge values calculated via finite difference (1 FMA)
- Perform local SIMD reduction in each cell

University of BRISTOL

Caches

Intel Xeon E5-2699v4 Processor (Broadwell)

- L1 and L2 per core, 32 KB and 256 KB
- L3 per socket, 55 MB

Intel Xeon Phi 7210 Processor (Knights Landing)

- L1 per core, 32 KB
- L2 per tile, 1 MB
- No L3

The mega-stream benchmark

University of BRISTOL

- Small benchmark code
- Easy to model (estimate) the memory bandwidth
- Distilled from the main kernel in SNAP mini-app
  - Performance proxy for a deterministic transport code
  - Motivation: Investigate performance issues of SNAP on Knights Landing

```
#pragma omp parallel for
for (int m = 0; m < Nm; m++) {
  for (int l = 0; l < Nl; l++) {
    for (int k = 0; k < Nk; k++) {
      for (int j = 0; j < Nj; j++) {
        double total = 0.0;
        #pragma omp simd reduction(+:total)
        for (int i = 0; i < Ni; i++) {
          /* Set r */
          r[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)] =
            q[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)] +
            a[i] * x[IDX4(i,j,k,m,Ni,Nj,Nk)] +
            b[i] * y[IDX4(i,j,l,m,Ni,Nj,Nl)] +
            c[i] * z[IDX4(i,k,l,m,Ni,Nk,Nl)];

          /* Update x, y and z */
          x[IDX4(i,j,k,m,Ni,Nj,Nk)] =
            0.2*r[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)] - x[IDX4(i,j,k,m,Ni,Nj,Nk)];

          y[IDX4(i,j,l,m,Ni,Nj,Nl)] =
            0.2*r[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)] - y[IDX4(i,j,l,m,Ni,Nj,Nl)];

          z[IDX4(i,k,l,m,Ni,Nk,Nl)] =
            0.2*r[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)] - z[IDX4(i,k,l,m,Ni,Nk,Nl)];

          /* Reduce over Ni */
          total += r[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)];

        } /* Ni */

        sum[IDX4(j,k,l,m,Nj,Nk,Nl)] += total;

      } /* Nj */
    } /* Nk */
  } /* Nl */
} /* Nm */
```

University of BRISTOL

Cell centre calculation

Outgoing face calculation

Within cell reduction

Experimental setup

- Intel® Xeon Phi™ 7210 Processor (Knights Landing)
  - DAP “Ninja” workstation
  - 1.30 GHz, 1.6 GHz mesh, 6.4 GT/s
  - 16 GB MCDRAM, Quad/Flat, 96 GB DDR (unused)
  - CentOS 7.2, XPPSL 1.5.1
  - Intel Compiler 17 update 2, `-O3 -xMIC-AVX512`
- Intel® Xeon® E5-2699 v4 Processor (Broadwell)
  - 22-cores, dual-socket, 2.2 GHz, 128 GB DDR
  - Cray XC40 node
  - Intel Compiler 17 update 1, `-O3 -xCORE-AVX2`

Default problem

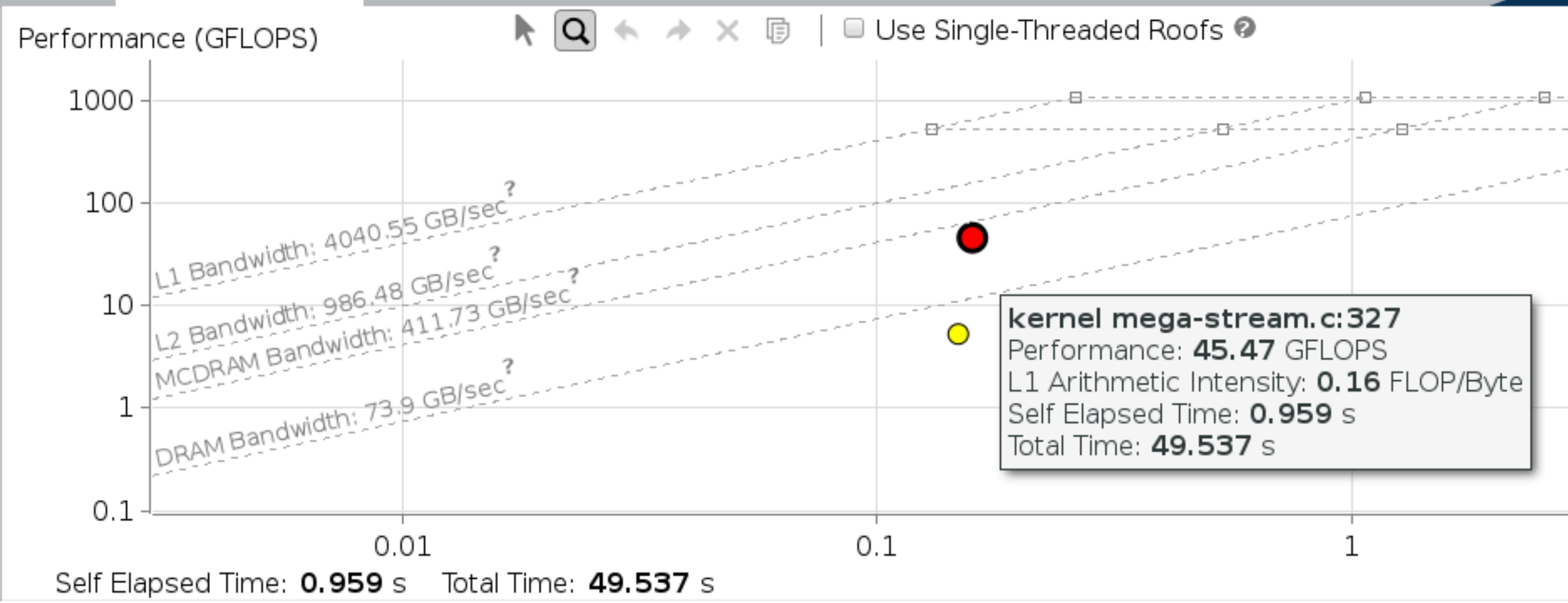
Ni	128
Nj, Nk, Nl	16
Nm	64

University of BRISTOL

Performance

	Broadwell		Knights Landing (MCDRAM)	
	Bandwidth	**% Triad**	**Bandwidth**	**% Triad**
Baseline	83 GB/s	65.1%	74 GB/s	16.4%

- Broadwell quite low, but not low enough to necessarily cause concern
- Knights Landing performance very low from MCDRAM
- But the code is already good!
  - Stride one access, vectorises, predictable access, etc



urce | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

: mega-stream.c:327 kernel

e	Source	Total Time	%
	const double * restrict b,		
	const double * restrict c,		
	double * restrict sum		
	)		
	{		
	#pragma omp parallel for	51.218s	
⊞	for (int m = 0; m < Nm; m++) {		
⊞	for (int l = 0; l < Nl; l++) {	0.020s	[
⊞	for (int k = 0; k < Nk; k++) {	0.020s	[
⊞	for (int j = 0; j < Nj; j++) {	0.260s	[
	double total = 0.0;	0.440s	[
	#pragma omp simd reduction(+:total)		
⊟	for (int i = 0; i < Ni; i++) {	0.840s	[
	[loop in kernel at mega-stream.c:327]		
	Vectorized AVX512F 512 loop processes Float32; Float64 data type(s) and includes FMA		

Improving performance

1. Ensure data which <u>is not</u> re-used <u>is not</u> in cache

   - Non-temporal stores

2. Ensure data which <u>is</u> re-used <u>is</u> in cache

   - Cache blocking

3. Ensure data is in the cache in time for use

   - Software prefetch

Non-temporal stores

- Store on Intel architectures typically does a “Read for Ownership”
- Reads from memory into cache, then writes to cache (and through to main memory)
- Streaming stores avoid read for ownership
- Prevents cache pollution

```
#pragma vector nontemporal(r)
```

University of BRISTOL

Performance

	Broadwell		Knights Landing (MCDRAM)	
	Bandwidth	**% Triad**	**Bandwidth**	**% Triad**
Baseline	83 GB/s	65.1%	74 GB/s	16.4%
Non-temporal stores	107 GB/s	83.7%	240 GB/s	53.6%

- 1.3X improvement on Broadwell
- 3X improvement on Knights Landing

University of BRISTOL

Cache blocking

- Reuse of x,y,z arrays, the incoming/outgoing edge arrays
- Want to be kept in cache (temporal locality)
- For default problem size, there arrays are 256 KiB per core
    - Total 768 KiB, but only 512 KB of L2 cache on Knights Landing
- Split inner (Ni) dimension into blocks of 8 (one cache line)
- Only need one cache line per array for j,k,l loops, which frees up cache

University of BRISTOL

Performance

	Broadwell		Knights Landing (MCDRAM)	
	Bandwidth	**% Triad**	**Bandwidth**	**% Triad**
Baseline	83 GB/s	65.1%	74 GB/s	16.4%
Non-temporal stores	107 GB/s	83.7%	240 GB/s	53.6%
Cache blocking	117 GB/s	91.8%	318 GB/s	71.0%

- 1.3X improvement on Knights Landing
- Edge arrays already fit in L3 cache on Broadwell, so only small improvement

Software prefetching

- Intel vTune Amplifier XE shows L2 cache misses for loading the q array (previous iteration cell centred values)
- Should just be streaming through this array, but hardware prefetcher not sufficient
- Turn on software prefetching (`-qopt-prefetch=3`) and see initial distance
- Add intrinsic and try distances until see improvement
- Switch to VLA syntax so compiler can calculate offset index

```
__mm_prefetch((const char *) &q[m][g][l][k][j][0] +
32*VLEN, _MM_HINT_T1);
```

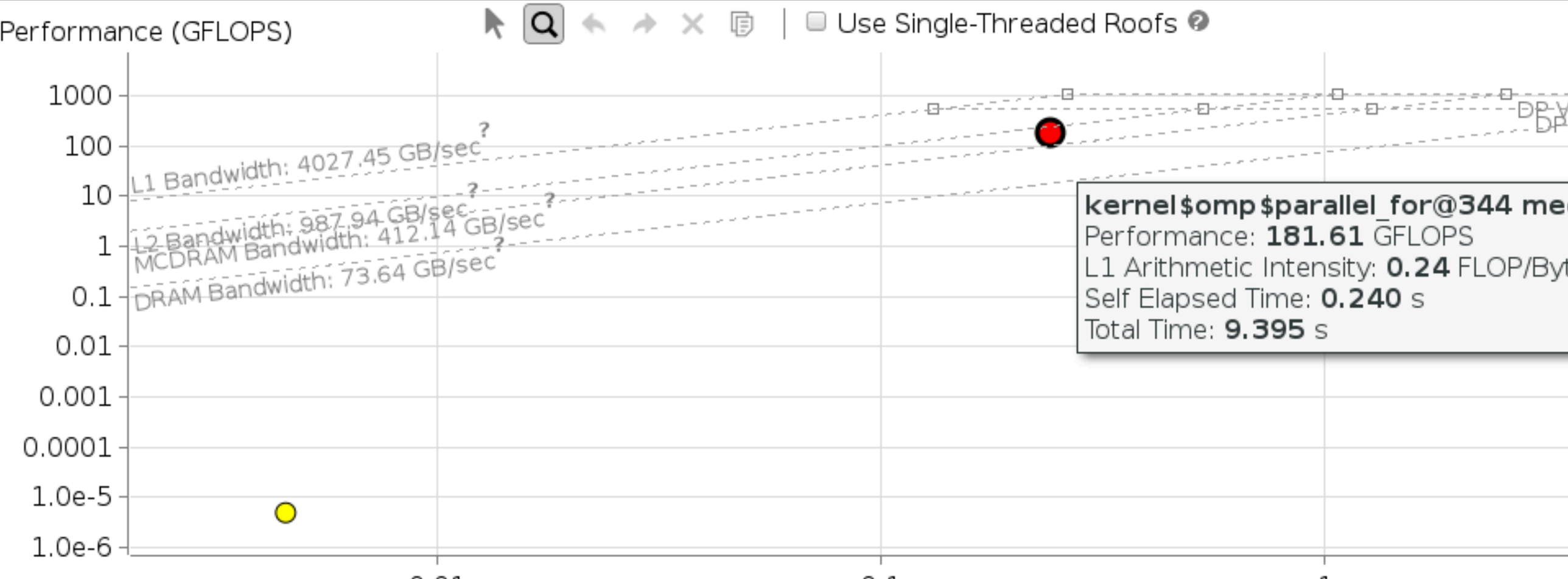
University of BRISTOL

Performance

	Broadwell		Knights Landing (MCDRAM)	
	Bandwidth	**% Triad**	**Bandwidth**	**% Triad**
Baseline	83 GB/s	65.1%	74 GB/s	16.4%
Non-temporal stores	107 GB/s	83.7%	240 GB/s	53.6%
Cache blocking	117 GB/s	91.8%	318 GB/s	71.0%
Software prefetch	109 GB/s	85.6%	349 GB/s	77.9%

- Hurts Broadwell, hardware prefetcher more sophisticated than Knights Landing

```
#pragma omp parallel for
for (int m = 0; m < Nm; m++) {
  for (int g = 0; g < Ng; g++) {
    for (int l = 0; l < Nl; l++) {
      for (int k = 0; k < Nk; k++) {
        for (int j = 0; j < Nj; j++) {
          double total = 0.0;
          _mm_prefetch((const char*) (&q[m][g][l][k][j][0] + 32*VLEN), _MM_HINT_T1);
          #pragma vector nontemporal(r)
          #pragma omp simd reduction(+:total) aligned(a,b,c,x,y,z,r,q:64)
          for (int v = 0; v < VLEN; v++) {
            /* Set r */
            r[m][g][l][k][j][v] =
              q[m][g][l][k][j][v] +
              a[g][v] * x[m][g][k][j][v] +
              b[g][v] * y[m][g][l][j][v] +
              c[g][v] * z[m][g][l][k][v];

            /* Update x, y and z */
            x[m][g][k][j][v] = 0.2*r[m][g][l][k][j][v] - x[m][g][k][j][v];
            y[m][g][l][j][v] = 0.2*r[m][g][l][k][j][v] - y[m][g][l][j][v];
            z[m][g][l][k][v] = 0.2*r[m][g][l][k][j][v] - z[m][g][l][k][v];

            /* Reduce over Ni */
            total += r[m][g][l][k][j][v];

          } /* VLEN */

          sum[m][l][k][j] += total;

        } /* Nj */
      } /* Nk */
    } /* Nl */
  } /* Ng */
} /* Nm */
```



Self Elapsed Time: **0.240** s Total Time: **9.395** s

ce | Top Down | Code Analytics | Assembly | Recommendations | Why No Vectorization?

mega-stream.c:349 kernelompparallel_for@344

Source	Total Time	%
y[IDX4(i,j,l,m,Ni,Nj,Nl)] = 0.2*tmp_r - y[IDX4(i,j,l,m,Ni,Nj,Nl)];		
z[IDX4(i,k,l,m,Ni,Nk,Nl)] = 0.2*tmp_r - z[IDX4(i,k,l,m,Ni,Nk,Nl)];		
/* Reduce over Ni */		
total += tmp_r;		
/* Save r */		
r[IDX5(i,j,k,l,m,Ni,Nj,Nk,Nl)] = tmp_r;	18.790s	
} /* Ni */		
sum[IDX4(j,k,l,m,Nj,Nk,Nl)] += total;		
[loop in kernelompparallel_for@344 at mega-stream.c:349]	2.304s	

University of BRISTOL

Summary

- Memory bandwidth bound kernels should be memory bandwidth bound
- Sometimes sensible, stride 1 access and vectorisation isn't enough if caches aren't being utilised as expected
    - Examine cache behaviour, back of envelope calculations help

References

- Mega-stream: https://github.com/UK-MAC/mega-stream
- SNAP: https://github.com/lanl/snap
- GPU SNAP publications:

[1] T. Deakin, S. McIntosh-Smith, M. Martineau, and W. Gaudin, “An improved parallelism scheme for deterministic discrete ordinates transport,” *Int. J. High Perform. Comput. Appl.*, Sep. 2016.

[2] T. Deakin, S. McIntosh-Smith, and W. Gaudin, “Many-Core Acceleration of a Discrete Ordinates Transport Mini-App at Extreme Scale,” in *High Performance Computing: 31st International Conference, ISC High Performance 2016, Frankfurt, Germany, June 19-23, 2016, Proceedings*, M. J. Kunkel, P. Balaji, and J. Dongarra, Eds. Cham: Springer International Publishing, 2016, pp. 429–448.