Abstract Visualization of Runtime Memory Behavior

A.N.M. Imroz Choudhury and Paul Rosen

Scientific Computing and Imaging Institute
School of Computing
University of Utah

September 29, 2011

Why Visualize Memory Behavior?

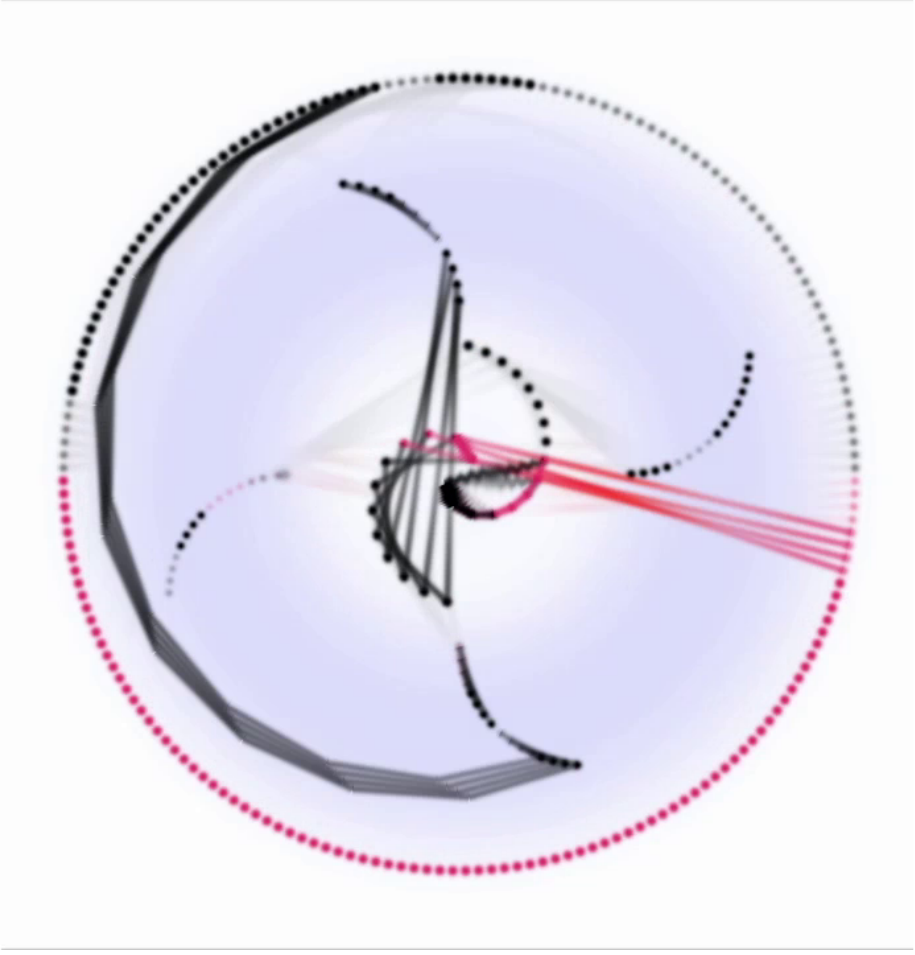
- Modern hardware + software → complex interactions
- e.g. caches: simple rules of operation, but complex behavior
- Sometimes performance is a design goal; cache performance is key to overall performance
  - Performance counters → coarse-grained information
  - Memory reference trace + cache simulation instead

Why Visualize Memory Behavior?

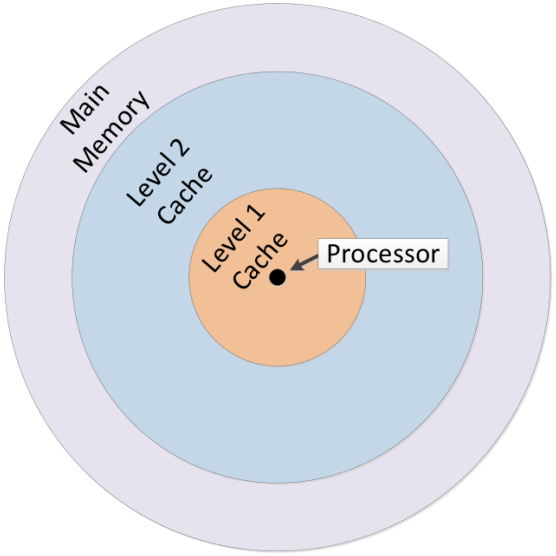
- Modern hardware + software → complex interactions
- e.g. caches: simple rules of operation, but complex behavior
- Sometimes performance is a design goal; cache performance is key to overall performance
  - Performance counters → coarse-grained information
  - Memory reference trace + cache simulation instead

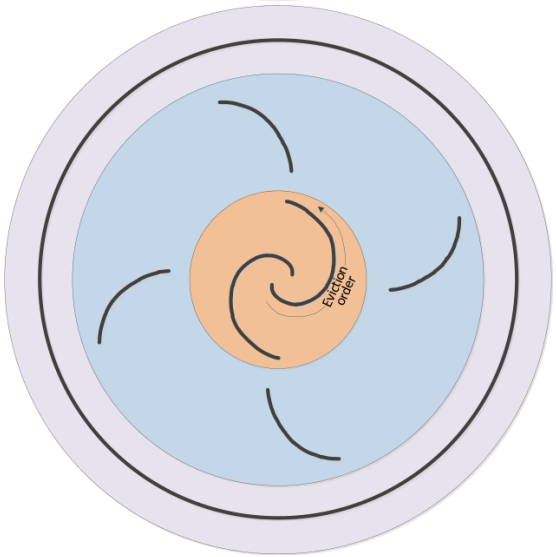
Our approach: simulate cache, visualize what happens inside, highlight performance issues, **move towards understanding memory performance**

Example



Overview of Approach

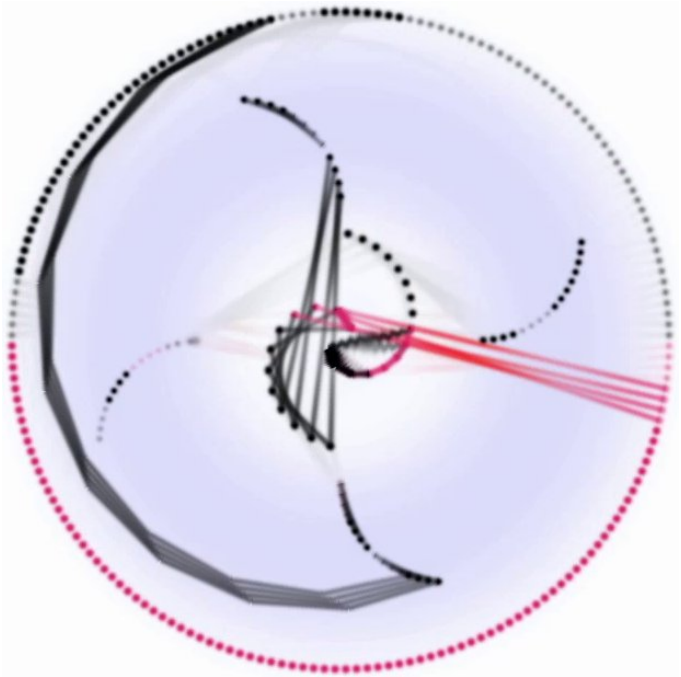




- Collect memory reference trace
  - Modify read/write instructions at run time
  - Trace includes source code information
- Cache simulation
  - Trace records (e.g. `R 0x400500f`) fed through simulator → hit or miss? eviction? etc.
- Visualization
  - Structural layout reflects cache architecture
  - Simulation results play out over layout

Visualization

Visual Channels and Frequencies

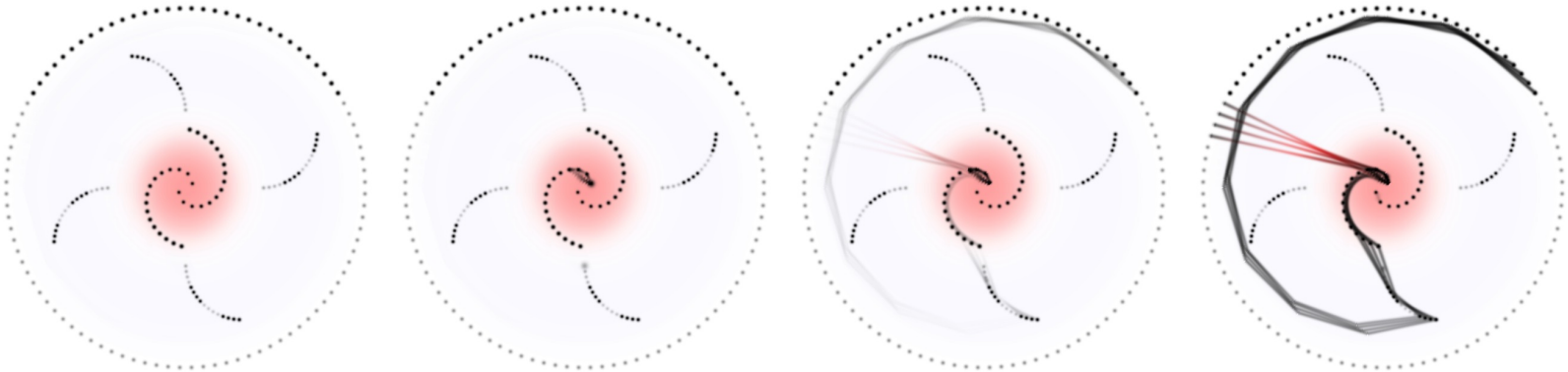


	Low frequency	**High frequency**
Color	home memory region	cache miss status
Motion	change in eviction order	change in cache level
Structure	associativity set	change in eviction order
Size	N/A	access

Visualization

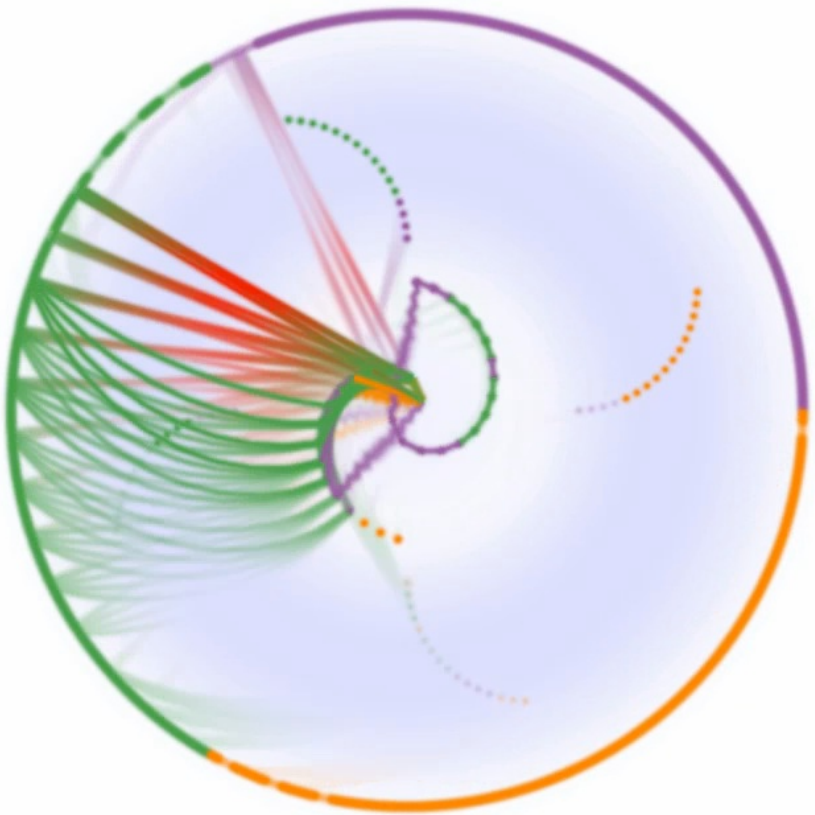
Time-Lapse Mode

- Very long traces, but can't just speed up playback
- Compress time → glyph *pathlines*
- Compute fewer such frames → time speeds up



Visualization

Time-Lapse Mode



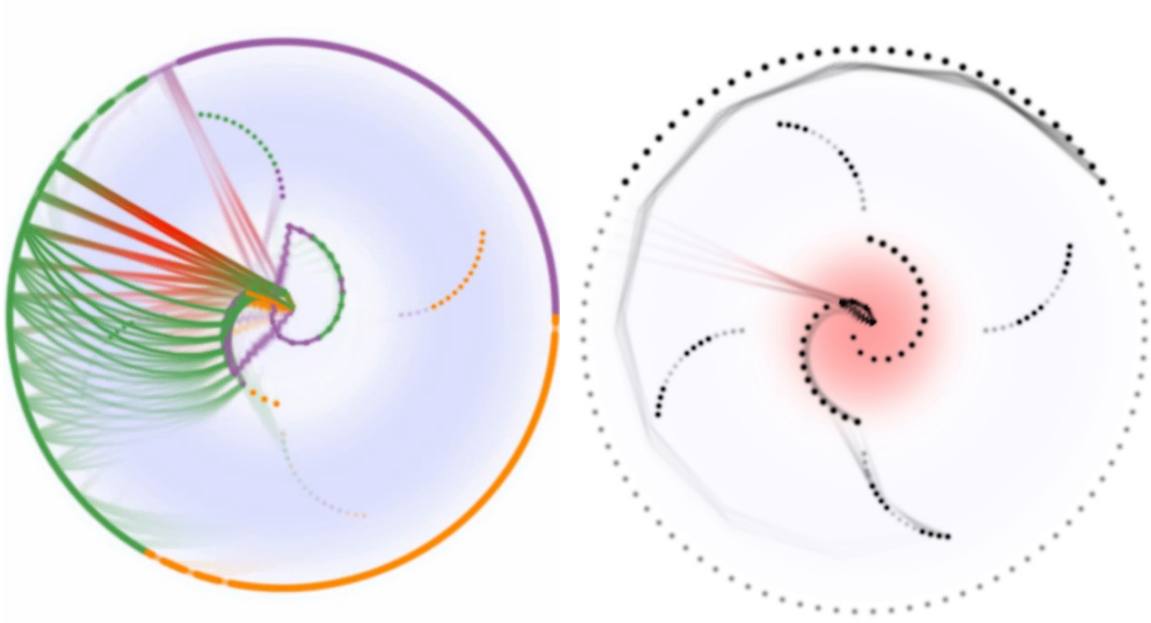
File: matmult.cpp

```
    case 6: x = &k; y = &j; z = &i; break;
    default: x = &i; y = &j; z = &k;
    }

  // Multiply
  for (*x = 0; *x < N; (*x)++)
    for (*y = 0; *y < N; (*y)++)
      for (*z = 0; *z < N; (*z)++)
        linC[i*N + j] += linA[i*N + k] * linB[k*N + j];

  return linC;
}

float** MatMult::mdIjk(unsigned int p)
{
  // Set up indices
  unsigned int i, j, k;
  unsigned int *x, *y, *z;

  switch (p)
    {
    case 1: x = &i; y = &j; z = &k; break;
    case 2: x = &i; y = &k; z = &j; break;
    case 3: x = &j; y = &i; z = &k; break;
    case 4: x = &j; y = &k; z = &i; break;
    case 5: x = &k; y = &i; z = &j; break;
    case 6: x = &k; y = &j; z = &i; break;
    default: x = &i; y = &j; z = &k;
    }

  // Multiply
  for (*x = 0; *x < N; (*x)++)
    for (*y = 0; *y < N; (*y)++)
      for (*z = 0; *z < N; (*z)++)
        mdC[i][j] += mdA[i][k] * mdB[k][j];

  return mdC;
}
```

Visualization

Summary View

- Expansive space behind data glyphs—display a summary statistic here
- "Cache temperature" summarizes recent cache performance, gives sense of history

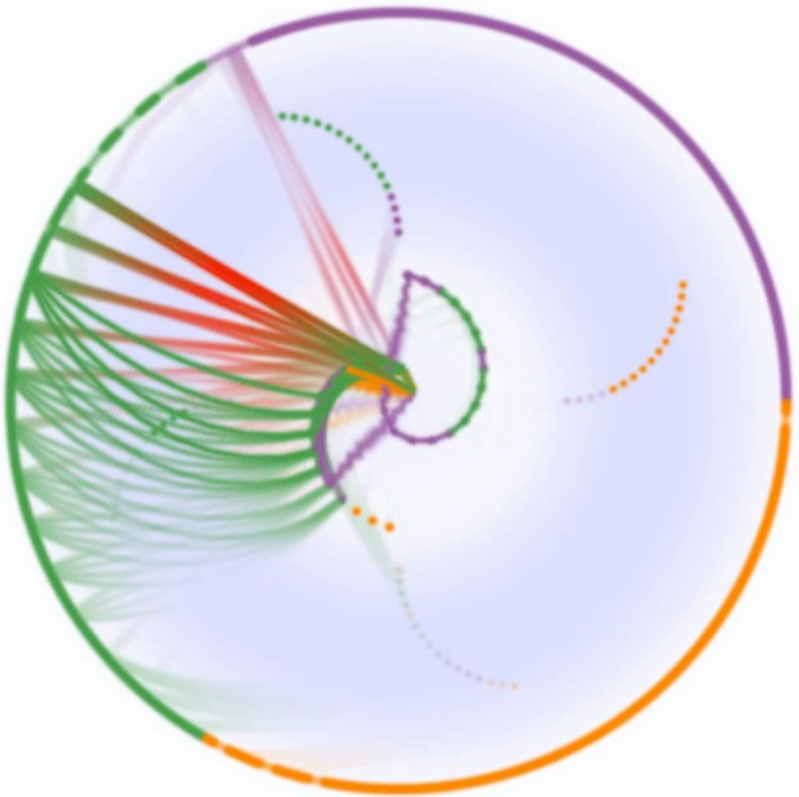


Standard Matrix Multiplication

- Left matrix: efficient access
- Right matrix: inefficient access

Standard Matrix Multiplication

$$A \times B = C$$

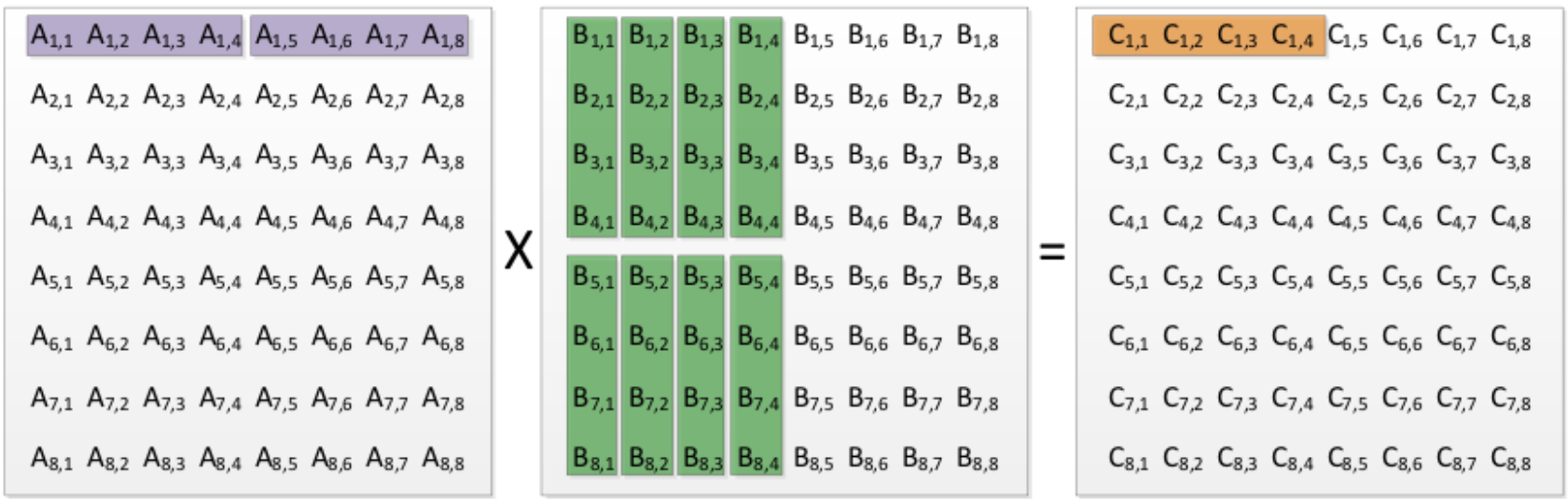


File: matmult.cpp

```
    case 6: x = &k; y = &j; z = &i; break;
    default: x = &i; y = &j; z = &k;
    }

  // Multiply
  for (*x = 0; *x < N; (*x)++)
    for (*y = 0; *y < N; (*y)++)
      for (*z = 0; *z < N; (*z)++)
        linC[i*N + j] += linA[i*N + k] * linB[k*N + j];

  return linC;
}

float** MatMult::mdIjk(unsigned int p)
{
  // Set up indices
  unsigned int i, j, k;
  unsigned int *x, *y, *z;

  switch (p)
    {
    case 1: x = &i; y = &j; z = &k; break;
    case 2: x = &i; y = &k; z = &j; break;
    case 3: x = &j; y = &i; z = &k; break;
    case 4: x = &j; y = &k; z = &i; break;
    case 5: x = &k; y = &i; z = &j; break;
    case 6: x = &k; y = &j; z = &i; break;
    default: x = &i; y = &j; z = &k;
    }

  // Multiply
  for (*x = 0; *x < N; (*x)++)
    for (*y = 0; *y < N; (*y)++)
      for (*z = 0; *z < N; (*z)++)
        mdC[i][j] += mdA[i][k] * mdB[k][j];

  return mdC;
}
```

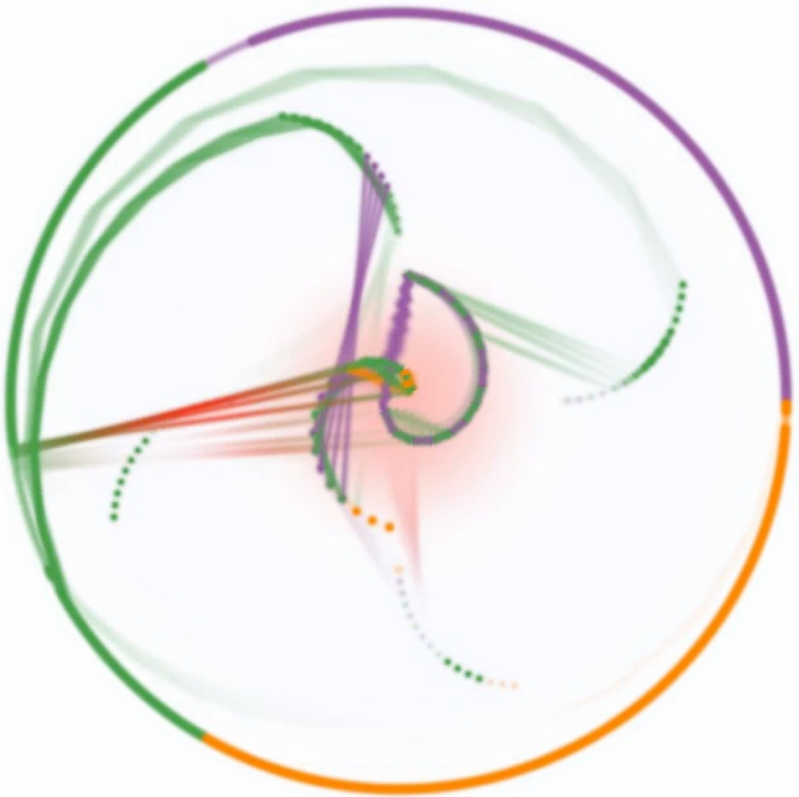
Transposed Matrix Multiplication

- Transpose right-hand matrix?
- Problem: loses generality



Transposed Matrix Multiplication

$$A \times B = C$$

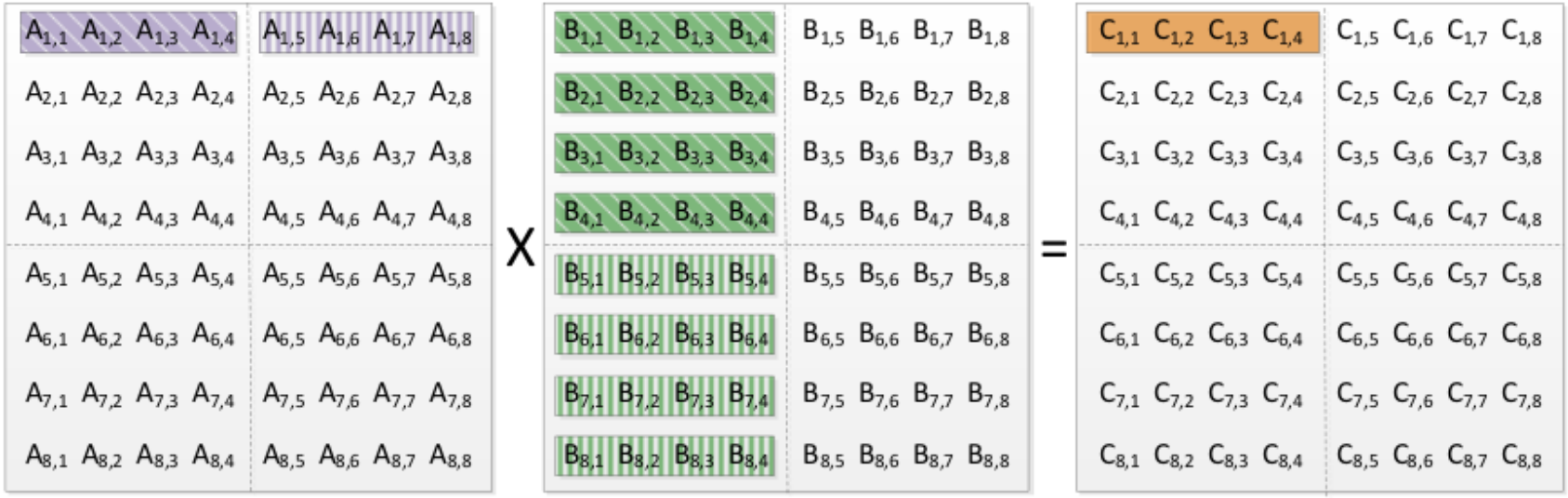


File: matmult-transpose.cpp

```
    case 6: x = &k; y = &j; z = &i; break;
    default: x = &i; y = &j; z = &k;
    }

  // Multiply
  for (*x = 0; *x < N; (*x)++)
    for (*y = 0; *y < N; (*y)++)
      for (*z = 0; *z < N; (*z)++)
    linC[i*N + j] += linA[i*N + k] * linB[j*N + k];

  return linC;
}

Real** MatMult::mdIjk(unsigned int p)
{
  // Set up indices
  unsigned int i, j, k;
  unsigned int *x, *y, *z;

  switch (p)
    {
    case 1: x = &i; y = &j; z = &k; break;
    case 2: x = &i; y = &k; z = &j; break;
    case 3: x = &j; y = &i; z = &k; break;
    case 4: x = &j; y = &k; z = &i; break;
    case 5: x = &k; y = &i; z = &j; break;
    case 6: x = &k; y = &j; z = &i; break;
    default: x = &i; y = &j; z = &k;
    }

  // Multiply
  for (*x = 0; *x < N; (*x)++)
    for (*y = 0; *y < N; (*y)++)
      for (*z = 0; *z < N; (*z)++)
    mdC[i][j] += mdA[i][k] * mdB[j][k];

  return mdC;
}
```

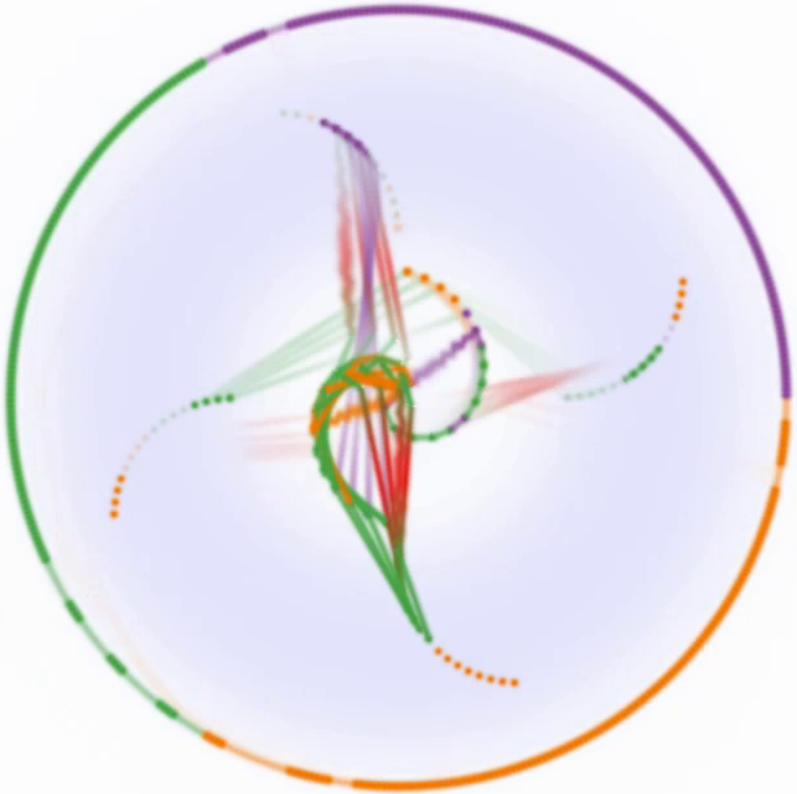
Blocked Matrix Multiplication

- Multiply small, cache-sized submatrices, accumulate into result
- Work on each cache line for longer $\rightarrow$ better data reuse



Blocked Matrix Multiplication

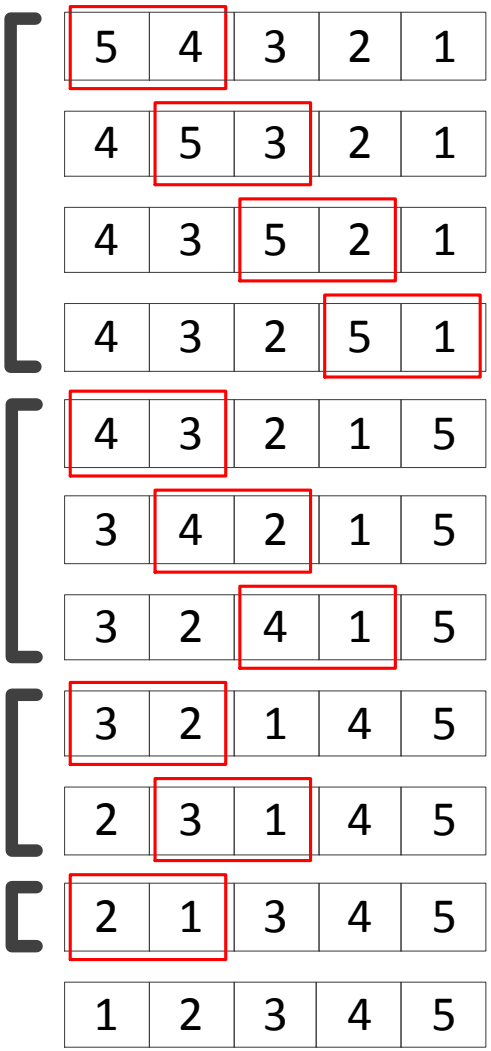
$$A \times B = C$$



File: matmult.cpp

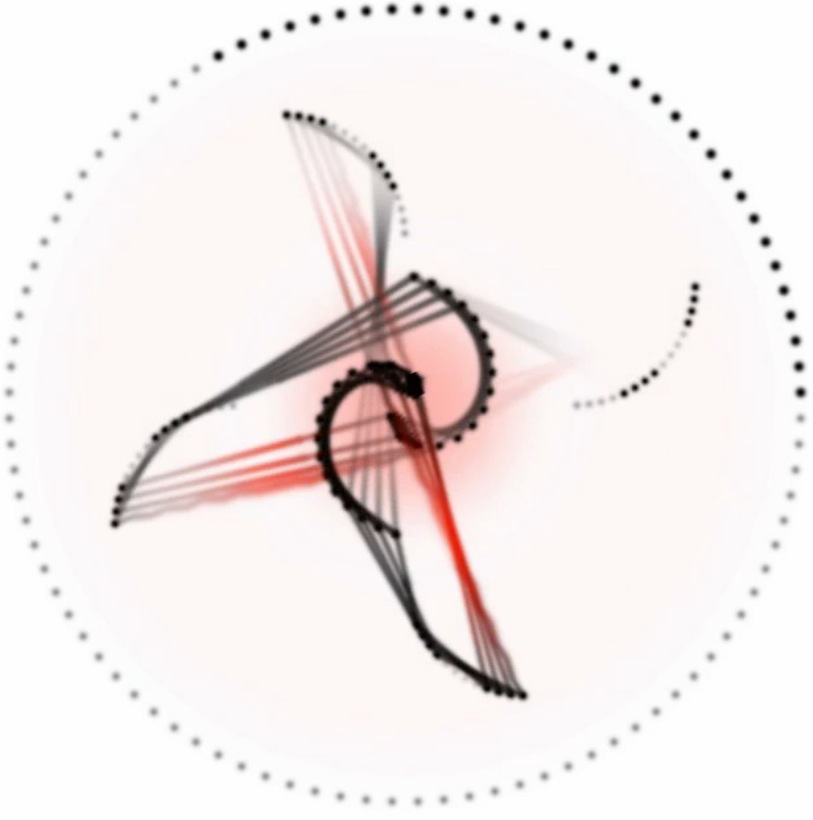
```
  for (unsigned int k0 = 0; k0 < N; k0 += b)
    for (unsigned int j0 = 0; j0 < N; j0 += b)
      for (unsigned int i = 0; i < N; i++)
        for (unsigned int k = k0; k < min(k0 + b, N); k++)
        {
          r = linA[i*N + k];
          for (unsigned int j = j0; j < min(j0 + b, N); j+
            linC[i*N + j] += r*linB[k*N + j];
        }

  return linC;
}

void MatMult::linRandM(float *A)
{
  for (unsigned int i = 0; i < N*N; i++)
    A[i] = ((float) rand()) / RAND_MAX;
}

void MatMult::mdRandM(float **A)
{
  for (unsigned int i = 0; i < N; i++)
      for (unsigned int j = 0; j < N; j++)
        A[i][j] = ((float) rand()) / RAND_MAX;
}

void MatMult::linPrintResults(float *A)
{
  printf("[");

  for (unsigned int i = 0; i < N*N; i++)
    {
      printf(" %f", A[i]);

      if ((i + 1) % N == 0)
        printf(";");
    }
```

Bubble Sort



- Sorts list by repeated, shrinking sweeps that swap one element to correct place
- Known for slow algorithmic complexity, $O(n^2)$
- Interesting cache performance—dramatic increase when working set fits into cache

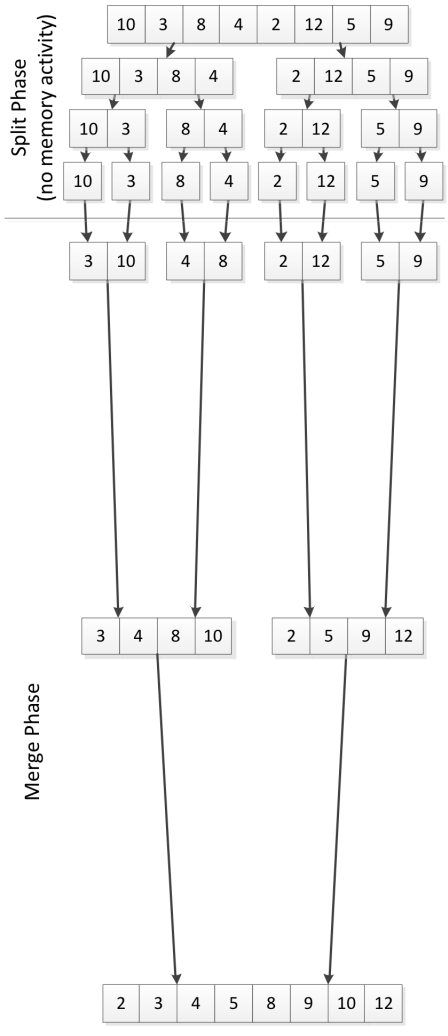
Bubble Sort



File: sort.cpp

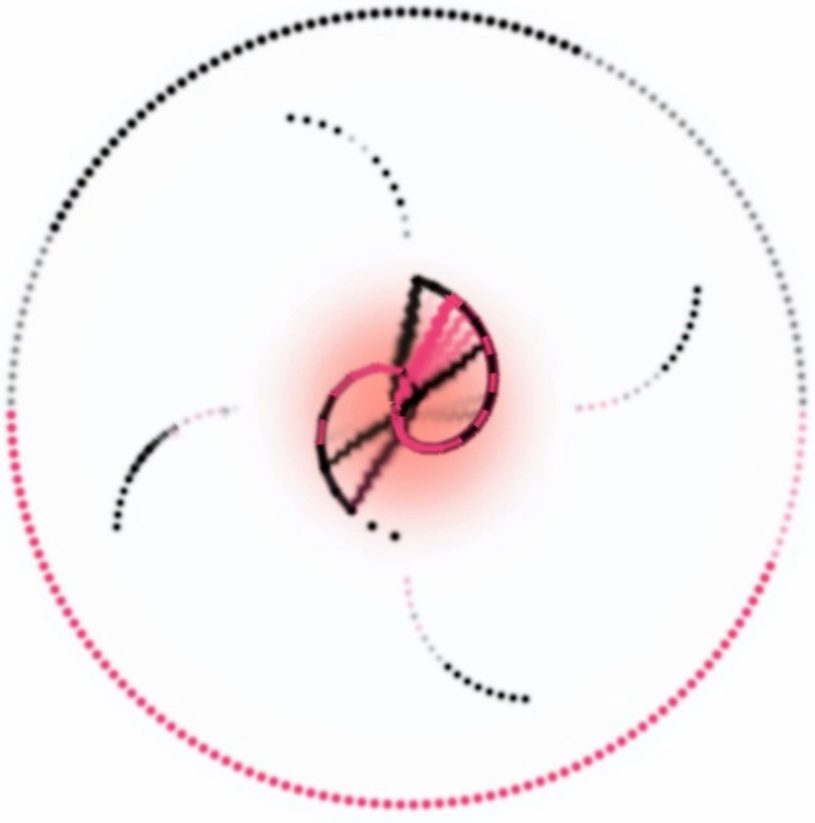
```
int floatCompare(const void *a, const void *b){
  return *(const float *)a - *(const float *)b;
}

void bubblesort(std::vector<float>& v){
  for(unsigned end=v.size()-1; end >= 0; end--){
    bool swapped = false;
    for(unsigned i=0; i<end; i++){
      if(v[i] > v[i+1]){
        std::swap(v[i], v[i+1]);
        swapped = true;
      }
    }
    if(!swapped){
      break;
    }
  }
}

int main(int argc, char *argv[]){
  // Get command line arguments.
  std::string inputfile, sortalgo;
  unsigned random;
  bool print, printbefore;

  try{
    // Create command line parser.
    TCLAP::CmdLine cmd("Sort a file's worth of numbers acc

    // The file containing the numbers to sort.
    TCLAP::ValueArg<std::string> inputfileArg("i",
                                              "input-file"
                                              "Newline-sepa
                                              false,
                                              "",
                                              "filename",
                                              cmd);
```

Merge Sort



- Better run time: $O(n \log n)$
- Recursive execution structure $\rightarrow$ more complex cache behavior

Merge Sort



File: mergesort.cpp

```
  // Merge sort sublists
  MergeSort( data, tmp, i0, i1 );
  MergeSort( data, tmp, i1, i2 );

  // Merge Lists
  int _i0 = i0;
  int _i1 = i1;
  for(int i = i0; i < i2; i++){
    if( _i1 >= i2 ){
      tmp[i] = data[_i0++];
    }
    else if( _i0 >= i1 ){
      tmp[i] = data[_i1++];
    }
    else if( data[_i0] < data[_i1] ){
      tmp[i] = data[_i0++];
    }
    else if( data[_i1] < data[_i0] ){
      tmp[i] = data[_i1++];
    }
  }

  // Copy data from temporary location
  for(int i = i0; i < i2; i++){
    data[i] = tmp[i];
  }

}

int main(int argc, char ** argv){
  srand(0);
        const int size = 96;
  vector<double> data(size);
  vector<double> tmp(size);

        MTR::Registrar r;
        r.array(reinterpret_cast<MTR::addr_t>(&data[0]), da
        r.array(reinterpret_cast<MTR::addr_t>(&tmp[0]), tmp
```

Material Point Method (MPM)

- Particle-based simulation method for mechanical engineering problems
- Particles carry several attributes, stored in several arrays
- Tends to show hallmark “streaming” access pattern

Material Point Method (MPM)

Conclusions

- Memory reference trace → Cache simulation → **Structured visualization of cache internals**
- Visual design assigns meaning to several visual channels
- Resulting visuals and patterns convey memory behavior

Future Work

- Exploration of unused or underused visual channels, e.g. low-frequency motion
- Expansion of techniques to other systems where performance is important

Thank you!