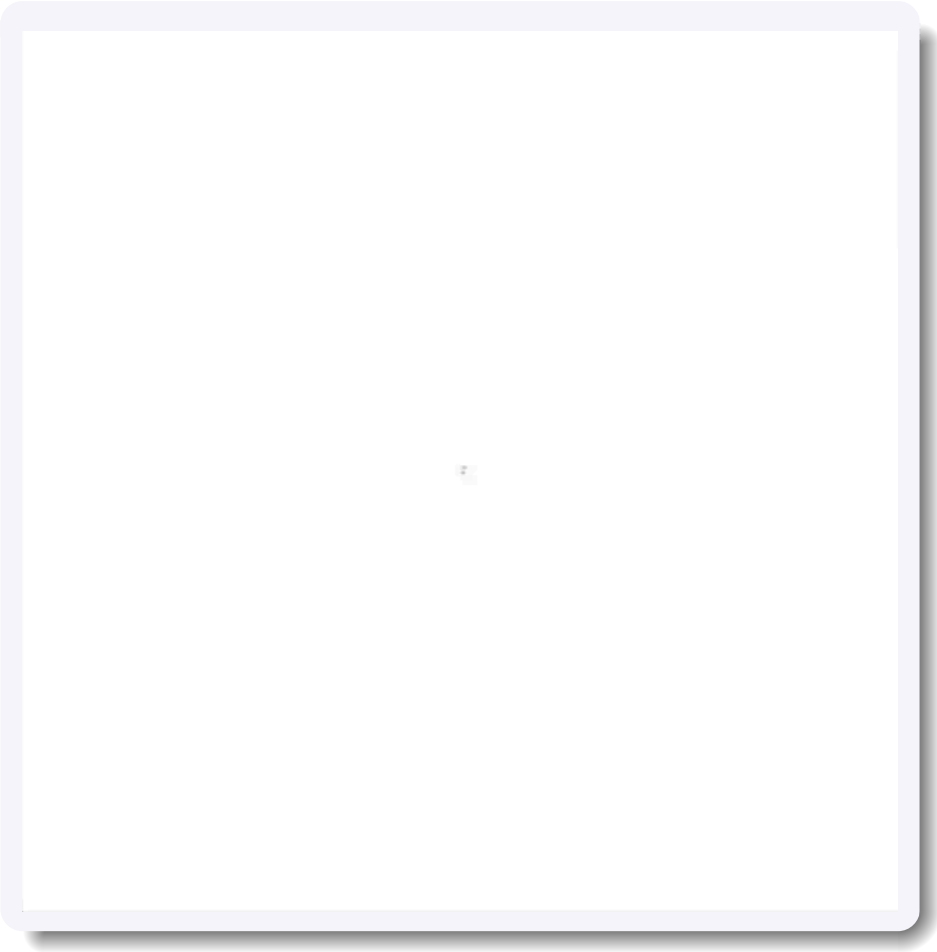
An optimization approach for the computational modeling of biological development

Pablo González de Aledo

July 2017

MC^2 Series

Agent-based Simulations



Courtesy of Roman Bauer (https://www.youtube.com/channel/UCE1DYTpHAhy_Ga-GoxYmUGw)
Newcastle University, UK. E-mail: roman.bauer@ncl.ac.uk

Behind Intel Modern Code Challenge

BioDynaMo

The Biology Dynamic Modeller

BioDynamo Project

- General platform for computer simulations of biological tissue dynamics.
- Efficient use of state-of-the-art computing technology.

Open Questions

- Techniques to improve agent-based simulations in HPC platforms?
- Compromises in readability, speedup, maintainability...?
- Partition the computational load in a hybrid cloud computing system?

Intel Modern Code Challenge

https://software.intel.com/en-us/modern-code
https://www.youtube.com/watch?v=pS0XUjWS73s

Goals

- Test the limits of Parallel Computing for biological simulation
- Introduce students to the work-flow of optimizing performance for HPC platforms
- Teach modernization techniques to biologist

Results

- Challenge was accepted by over 17.000 students.
- more than 130 universities.
- 19 countries.
- 320x speedup.

Intel Modern Code Challenge

https://software.intel.com/en-us/modern-code
https://www.youtube.com/watch?v=pS0XUjWS73s

Goals

- Test the limits of Parallel Computing for biological simulation
- Introduce students to the work-flow of optimizing performance for HPC platforms
- Teach modernization techniques to biologist

Results

About this talk

Get-your-hands-dirty approach

- Source code will be available after the presentation

Agenda

- Introduction
  - Used Intel architectures
  - How to replicate our results
- Optimizations
  - Parallel pattern identification
  - Parallelization
  - Vectorization
  - Loop reordering and fusion
  - Map-reduce optimizations
  - Memory management
  - Compiler Optimizations
- Conclusion

Gratitude

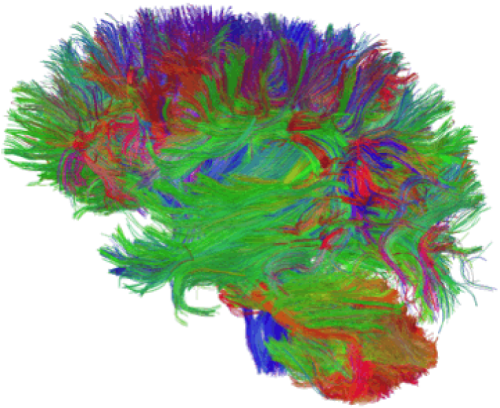
Newcastle University

Dr Roman Bauer
MRC Fellow

Institute of Neuroscience
Newcastle University

Prof. Marcus Kaiser
Professor of Neuroinformatics

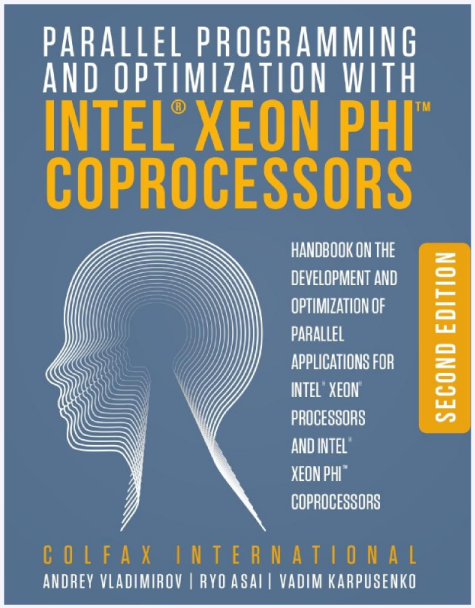
School of Computing Science
Newcastle University



Colfax Research

Dr Andrey Vladimirov
Head of HPC Research

Colfax International
North Carolina State University

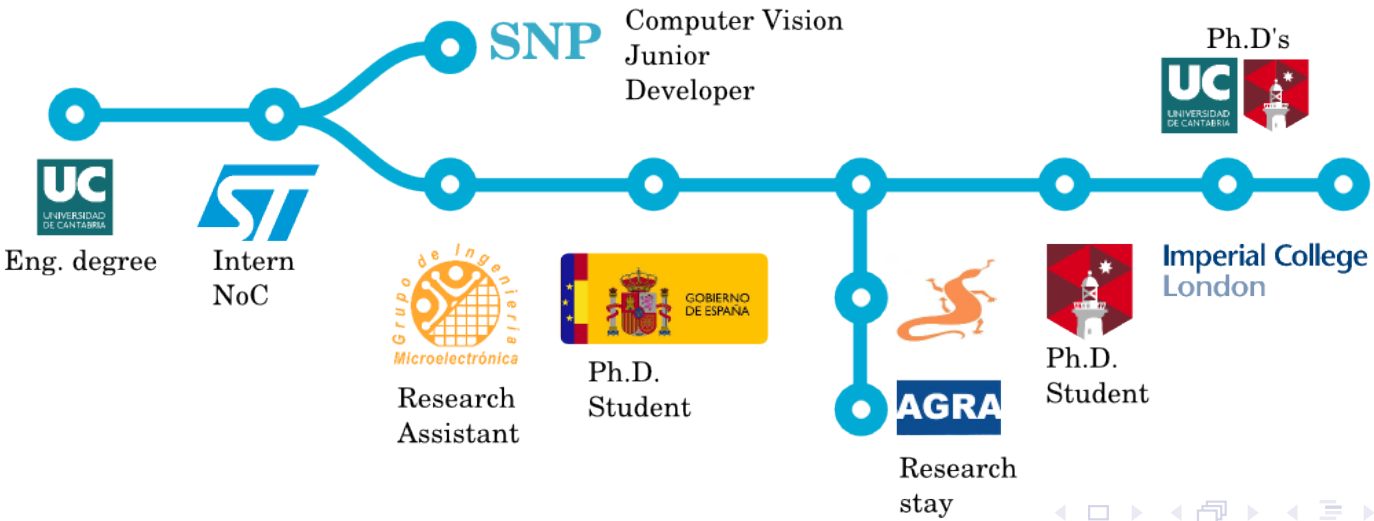


About me

- Ph.D. Candidate at Cantabria and MacQuarie Universities.
- http://apt.cs.manchester.ac.uk/projects/PAMELA/

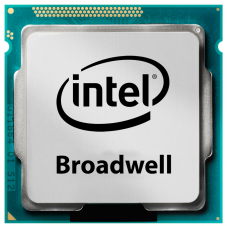
Interests

- Simulation of embedded platforms.
- Verification of software properties.
- HPC and power efficiency.
- Languages and compilers for HPC.



Computational Platforms

- Intel Xeon processor E5-2690
- 2 packages with 8 cores each @ 2.90 GHz
- 64 GB of DDR4 memory at 1333 MHz
- 20 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX instructions



- Intel Xeon processor E5-2699 v4
- 2 packages with 22 cores @ 2.20 GHz
- 128 GB of DDR4 memory
- 55 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX and AVX2 instructions

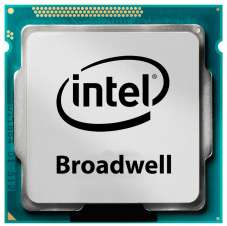
- Intel Xeon Phi processor 7250
- 68 cores @ 1.40 GHz
- 94 GB of DDR4 memory
- 16 GB of L3, 1 MB of L2, 32 KB of L1
- AVX-512 instructions

Considerations

-

Computational Platforms

- Intel Xeon processor E5-2690
- 2 packages with 8 cores each @ 2.90 GHz
- 64 GB of DDR4 memory at 1333 MHz
- 20 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX instructions



- Intel Xeon processor E5-2699 v4
- 2 packages with 22 cores @ 2.20 GHz
- 128 GB of DDR4 memory
- 55 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX and AVX2 instructions

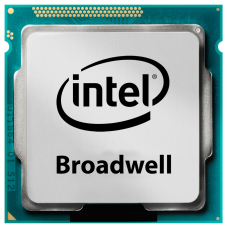
- Intel Xeon Phi processor 7250
- 68 cores @ 1.40 GHz
- 94 GB of DDR4 memory
- 16 GB of L3, 1 MB of L2, 32 KB of L1
- AVX-512 instructions

Considerations

- Use all processors

Computational Platforms

- Intel Xeon processor E5-2690
- 2 packages with 8 cores each @ 2.90 GHz
- 64 GB of DDR4 memory at 1333 MHz
- 20 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX instructions



- Intel Xeon processor E5-2699 v4
- 2 packages with 22 cores @ 2.20 GHz
- 128 GB of DDR4 memory
- 55 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX and AVX2 instructions

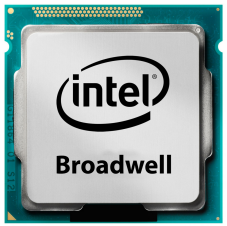
- Intel Xeon Phi processor 7250
- 68 cores @ 1.40 GHz
- 94 GB of DDR4 memory
- 16 GB of L3, 1 MB of L2, 32 KB of L1
- AVX-512 instructions

Considerations

- Use all processors
- Vectorize

Computational Platforms

- Intel Xeon processor E5-2690
- 2 packages with 8 cores each @ 2.90 GHz
- 64 GB of DDR4 memory at 1333 MHz
- 20 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX instructions



- Intel Xeon processor E5-2699 v4
- 2 packages with 22 cores @ 2.20 GHz
- 128 GB of DDR4 memory
- 55 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX and AVX2 instructions

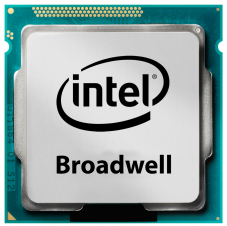
- Intel Xeon Phi processor 7250
- 68 cores @ 1.40 GHz
- 94 GB of DDR4 memory
- 16 GB of L3, 1 MB of L2, 32 KB of L1
- AVX-512 instructions

Considerations

- Use all processors
- Vectorize
- Consider memory access patterns

Computational Platforms

- Intel Xeon processor E5-2690
- 2 packages with 8 cores each @ 2.90 GHz
- 64 GB of DDR4 memory at 1333 MHz
- 20 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX instructions



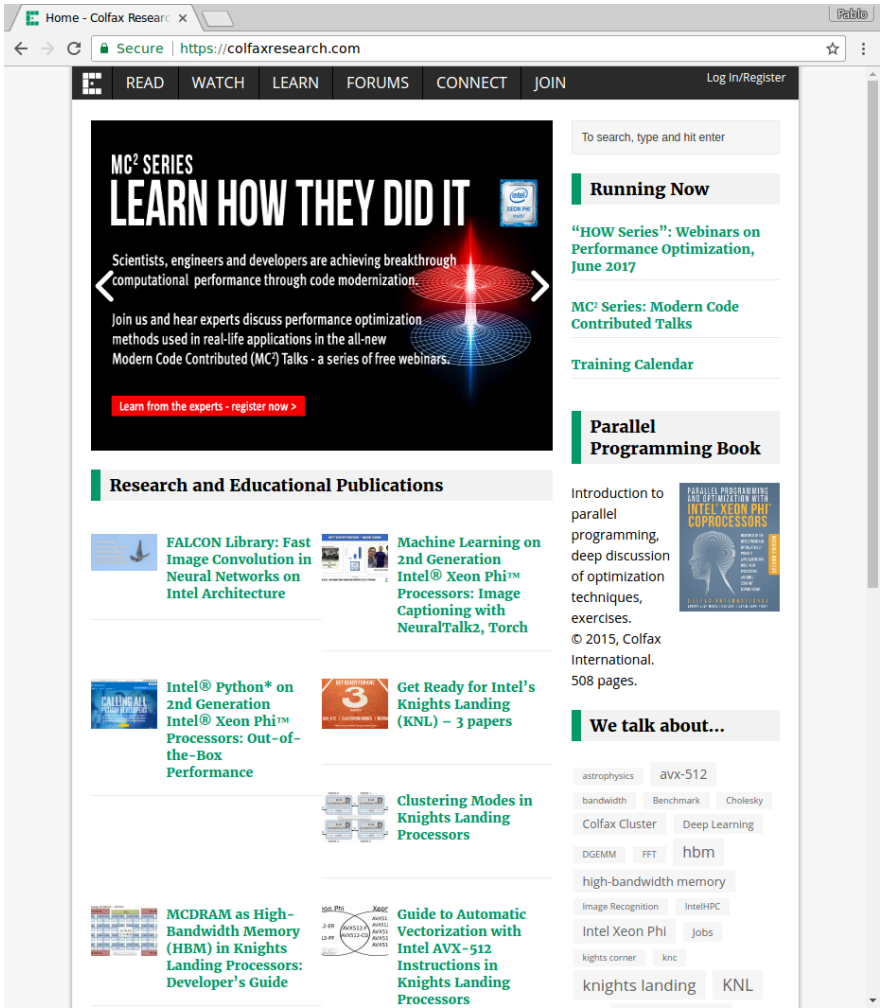
- Intel Xeon processor E5-2699 v4
- 2 packages with 22 cores @ 2.20 GHz
- 128 GB of DDR4 memory
- 55 MB of L3, 256 KB of L2, 32 KB of L1
- 256-bit AVX and AVX2 instructions

- Intel Xeon Phi processor 7250
- 68 cores @ 1.40 GHz
- 94 GB of DDR4 memory
- 16 GB of L3, 1 MB of L2, 32 KB of L1
- AVX-512 instructions

Considerations

- Use all processors
- Vectorize
- Consider memory access patterns
- Consider cache efficiency

Replicating our results

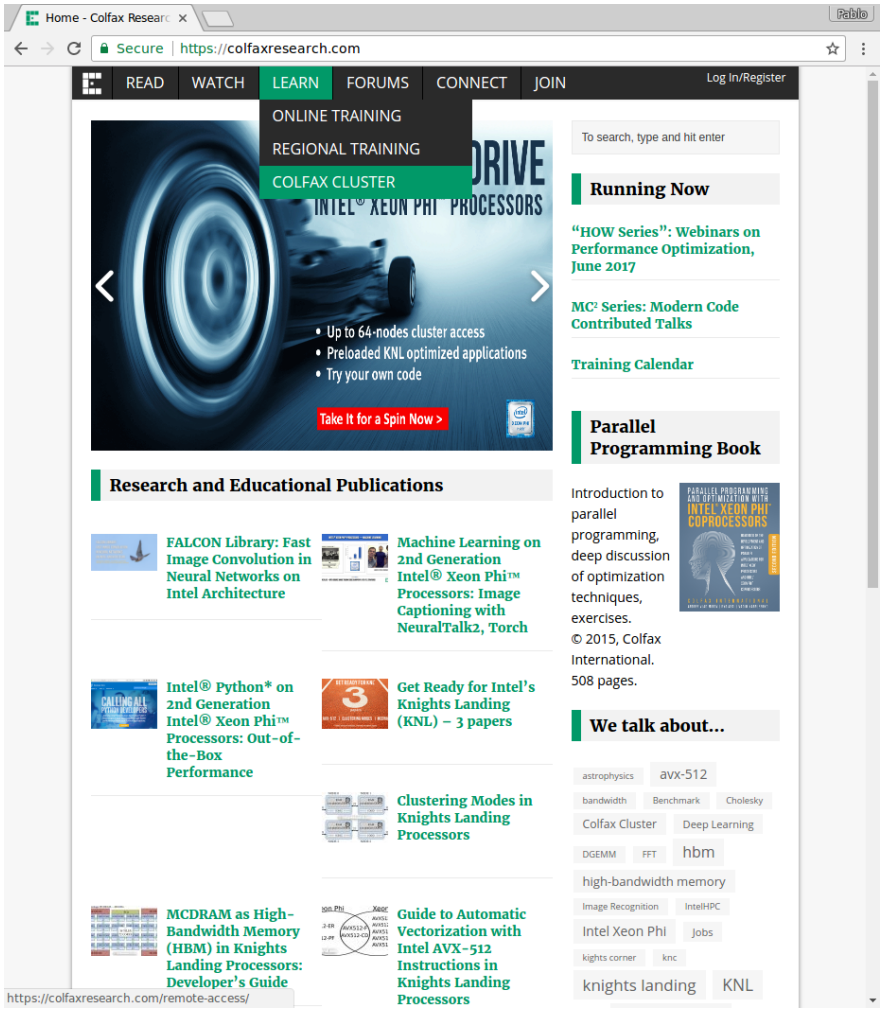



```
ssh colfax
```

```
make
make run-knl
make run-snb
make run-bdw
```

```
PATHTHRESHOLD                       = 2.000000e+00
DIVTHRESHOLD                        = 16
-----------------------------------------
INITIALIZATION_TIME                 = 3.710384e-02 s
PHASE1_TIME                         = 2.714862e+00 s
INITIAL_CRITERION                   = 0
INITIAL_ENERGY                      = 7.071191e-05
FINAL_CRITERION                     = 1
FINAL_ENERGY                        = -8.682492e-02
PHASE2_TIME                         = 1.577081e+01 s
produceSubstances_TIME              = 3.247619e+00 s (17.57 %)
runDiffusionStep_TIME               = 5.251892e+00 s (28.41 %)
runDecayStep_TIME                   = 3.751182e-01 s (2.03 %)
cellMovementAndDuplication_TIME     = 2.081288e-01 s (1.13 %)
runDiffusionClusterStep_TIME        = 6.559357e-01 s (3.55 %)
getEnergy_TIME                      = 3.721332e+00 s (20.13 %)
getCriterion_TIME                   = 3.714156e+00 s (20.09 %)
extra_TIME                          = 0.000000e+00 s (0.00 %)
TOTAL_COMPUTE_TIME                  = 1.848570e+01 s (100.00 %)
=================================================
```

Replicating our results

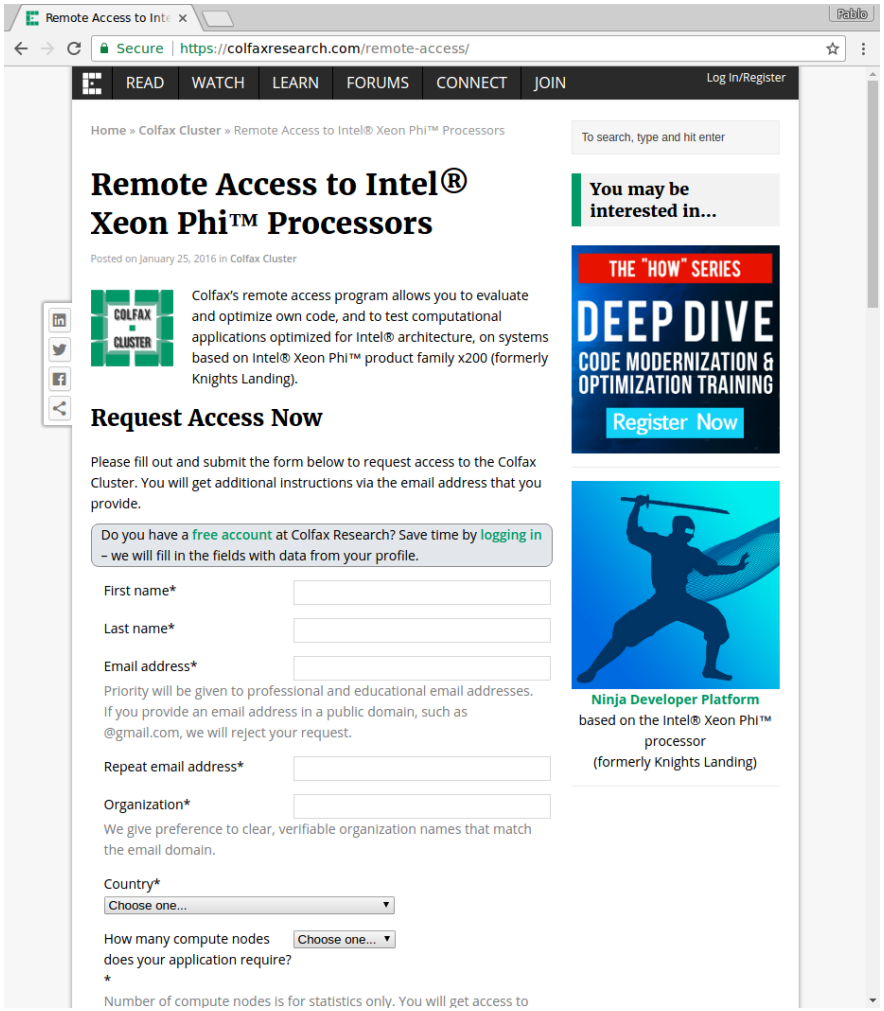



```
ssh colfax
```

```
make
make run-knl
make run-snb
make run-bdw
```

```
PATHTHRESHOLD                       = 2.000000e+00
DIVTHRESHOLD                        = 16
-----------------------------------------
INITIALIZATION_TIME                 = 3.710384e-02 s
PHASE1_TIME                         = 2.714862e+00 s
INITIAL_CRITERION                   = 0
INITIAL_ENERGY                      = 7.071191e-05
FINAL_CRITERION                     = 1
FINAL_ENERGY                        = -8.682492e-02
PHASE2_TIME                         = 1.577081e+01 s
produceSubstances_TIME              = 3.247619e+00 s (17.57 %)
runDiffusionStep_TIME               = 5.251892e+00 s (28.41 %)
runDecayStep_TIME                   = 3.751182e-01 s (2.03 %)
cellMovementAndDuplication_TIME     = 2.081288e-01 s (1.13 %)
runDiffusionClusterStep_TIME        = 6.559357e-01 s (3.55 %)
getEnergy_TIME                      = 3.721332e+00 s (20.13 %)
getCriterion_TIME                   = 3.714156e+00 s (20.09 %)
extra_TIME                          = 0.000000e+00 s (0.00 %)
TOTAL_COMPUTE_TIME                  = 1.848570e+01 s (100.00 %)
=================================================
```

Replicating our results

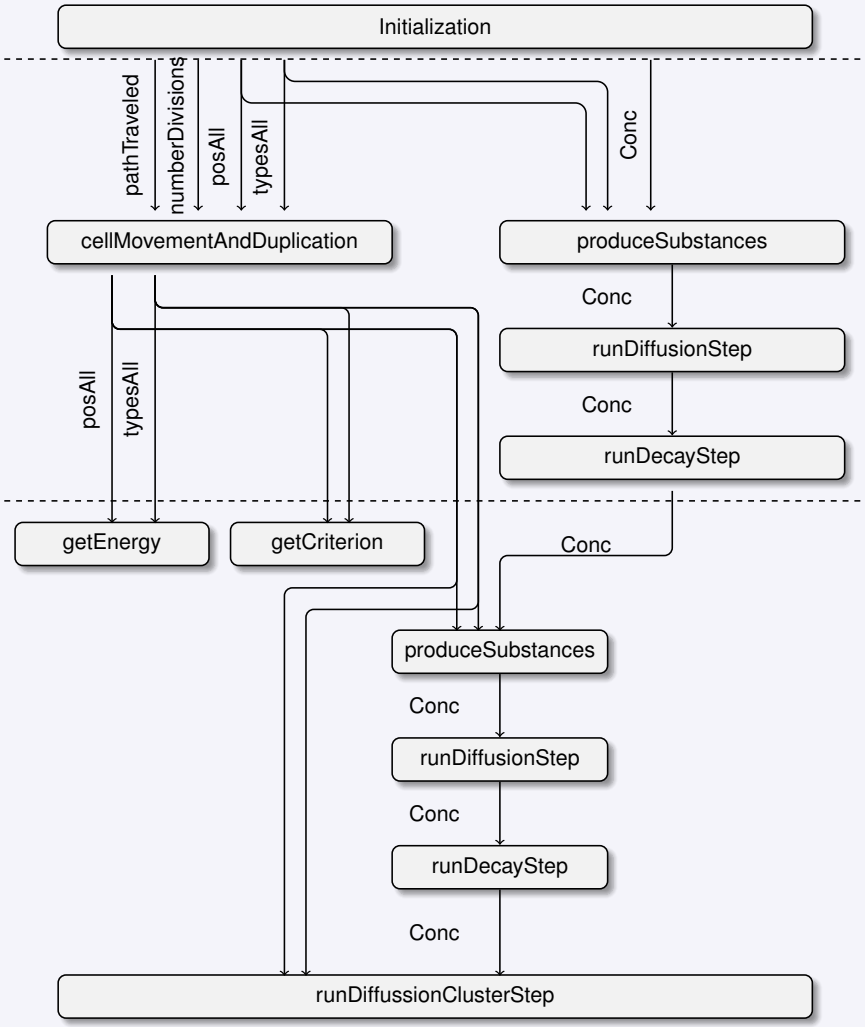



```
ssh colfax
```

```
make
make run-knl
make run-snb
make run-bdw
```

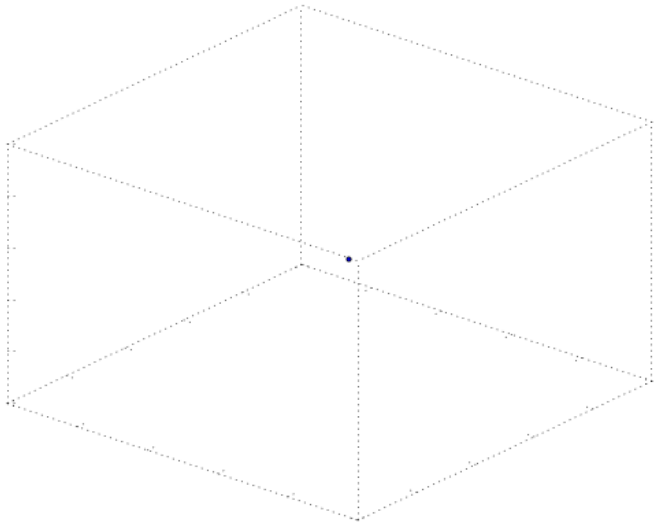
```
PATHTHRESHOLD                       = 2.000000e+00
DIVTHRESHOLD                        = 16
-----------------------------------------
INITIALIZATION_TIME                 = 3.710384e-02 s
PHASE1_TIME                         = 2.714862e+00 s
INITIAL_CRITERION                   = 0
INITIAL_ENERGY                      = 7.071191e-05
FINAL_CRITERION                     = 1
FINAL_ENERGY                        = -8.682492e-02
PHASE2_TIME                         = 1.577081e+01 s
produceSubstances_TIME              = 3.247619e+00 s (17.57 %)
runDiffusionStep_TIME               = 5.251892e+00 s (28.41 %)
runDecayStep_TIME                   = 3.751182e-01 s (2.03 %)
cellMovementAndDuplication_TIME     = 2.081288e-01 s (1.13 %)
runDiffusionClusterStep_TIME        = 6.559357e-01 s (3.55 %)
getEnergy_TIME                      = 3.721332e+00 s (20.13 %)
getCriterion_TIME                   = 3.714156e+00 s (20.09 %)
extra_TIME                          = 0.000000e+00 s (0.00 %)
TOTAL_COMPUTE_TIME                  = 1.848570e+01 s (100.00 %)
=================================================
```

Source Code Initial Architecture



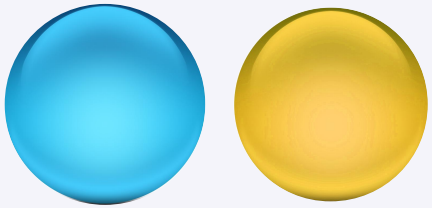
2 Phases

- Initial cells creation
- Cell clustering



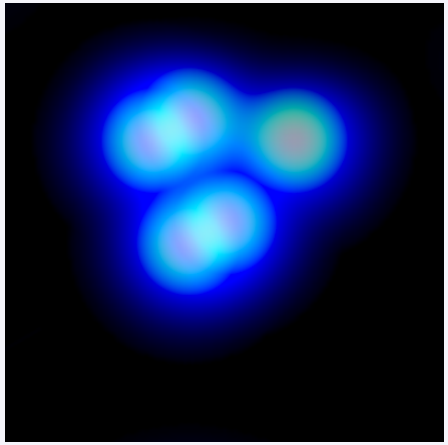
Moving pieces

cells



- move
- know how much they have travelled
- ‘sense’ chemicals
- divide
- cluster
- produce two kind of substances

substances



- increase/decrease
- move to neighbouring voxels

Code Modernization

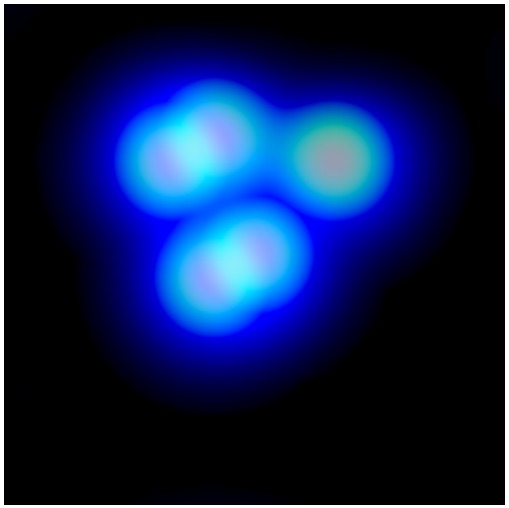
Overall Approach

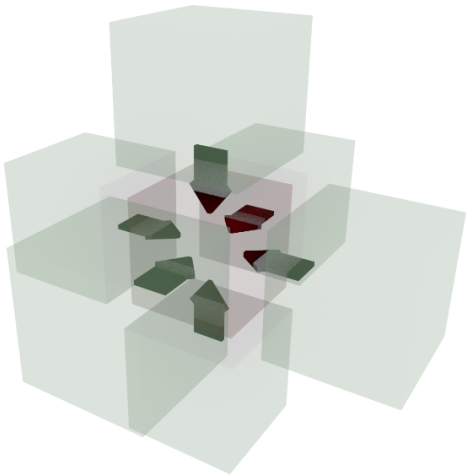
- Instrumentation
- Identification of Parallel Patterns
- Optimization

Obstacles to parallelization

- Inter-procedural analysis
- Pointer Reasoning
- Reliable identification of patterns

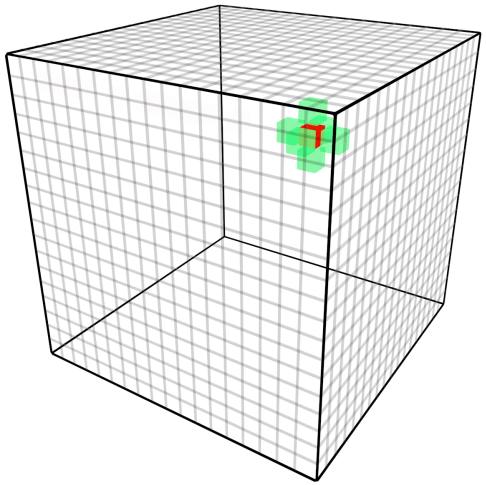
Parallelization of runDiffusionStep

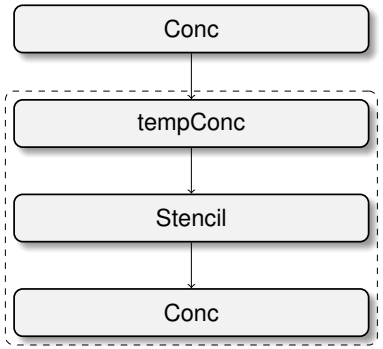


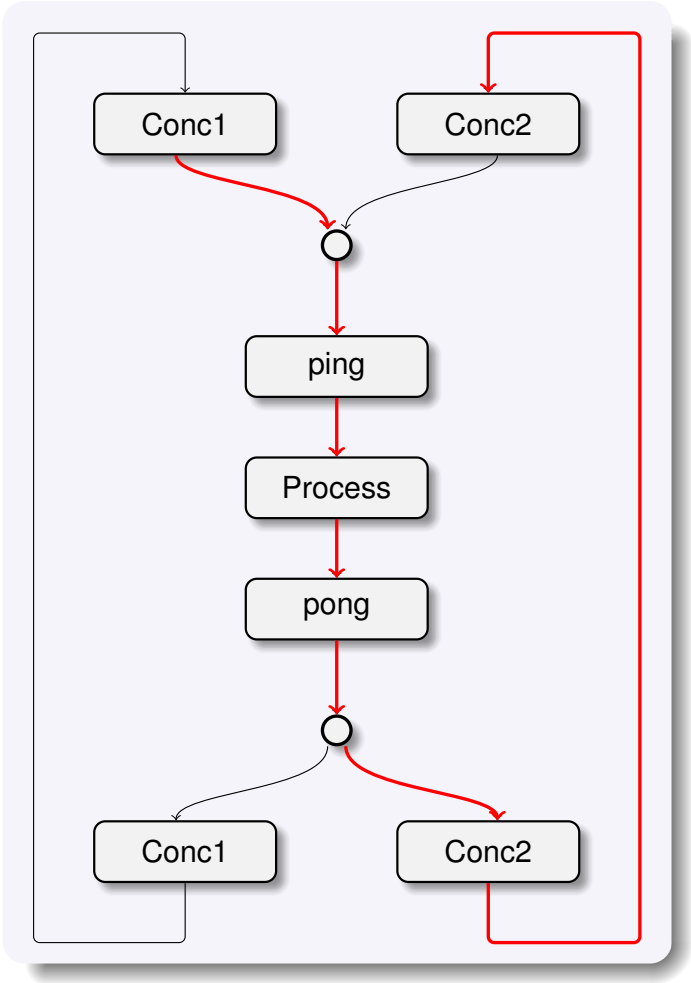


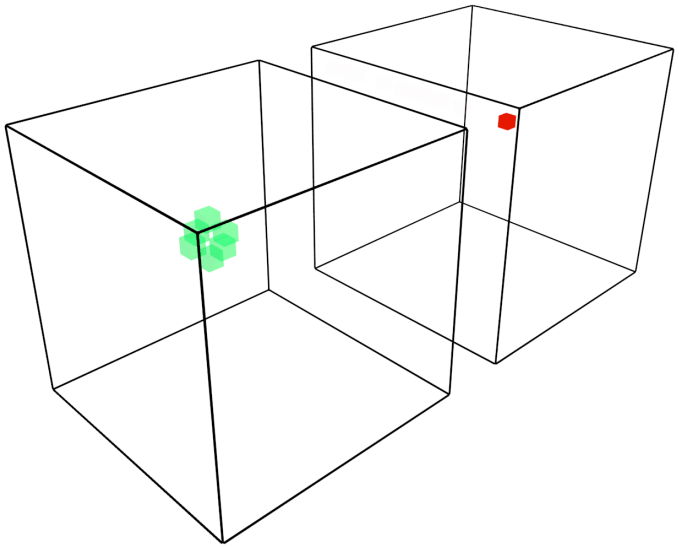
```
static void runDiffusionStep(float**** Conc, int L, float D) {
    runDiffusionStep_sw.reset();
    // computes the changes in substance concentrations due to diffusion
    int i1,i2,i3, subInd;
    float tempConc[2][L][L][L];
    for (i1 = 0; i1 < L; i1++) {
        for (i2 = 0; i2 < L; i2++) {
            for (i3 = 0; i3 < L; i3++) {
                tempConc[0][i1][i2][i3] = Conc[0][i1][i2][i3];
                tempConc[1][i1][i2][i3] = Conc[1][i1][i2][i3];
            }
        }
    }

    int xUp, xDown, yUp, yDown, zUp, zDown;

    for (i1 = 0; i1 < L; i1++) {
        for (i2 = 0; i2 < L; i2++) {
            for (i3 = 0; i3 < L; i3++) {
                xUp = (i1+1);
                xDown = (i1-1);
                yUp = (i2+1);
                yDown = (i2-1);
                zUp = (i3+1);
                zDown = (i3-1);

                for (subInd = 0; subInd < 2; subInd++) {
                    if (xUp<L) {
                        Conc[subInd][i1][i2][i3] += (tempConc[subInd][xUp][i2][i3]-tempConc[subInd][i1][i2][i3])*D/6;
                    }
                    if (xDown>=0) {
                        Conc[subInd][i1][i2][i3] += (tempConc[subInd][xDown][i2][i3]-tempConc[subInd][i1][i2][i3])*D/6;
                    }
                    if (yUp<L) {
                        Conc[subInd][i1][i2][i3] += (tempConc[subInd][i1][yUp][i3]-tempConc[subInd][i1][i2][i3])*D/6;
                    }
                    if (yDown>=0) {
                        Conc[subInd][i1][i2][i3] += (tempConc[subInd][i1][yDown][i3]-tempConc[subInd][i1][i2][i3])*D/6;
                    }
                    if (zUp<L) {
                        Conc[subInd][i1][i2][i3] += (tempConc[subInd][i1][i2][zUp]-tempConc[subInd][i1][i2][i3])*D/6;
                    }
                    if (zDown>=0) {
                        Conc[subInd][i1][i2][i3] += (tempConc[subInd][i1][i2][zDown]-tempConc[subInd][i1][i2][i3])*D/6;
                    }
                }
            }
        }
    }
    runDiffusionStep_sw.mark();
}
```

Optimization of runDiffusionStep

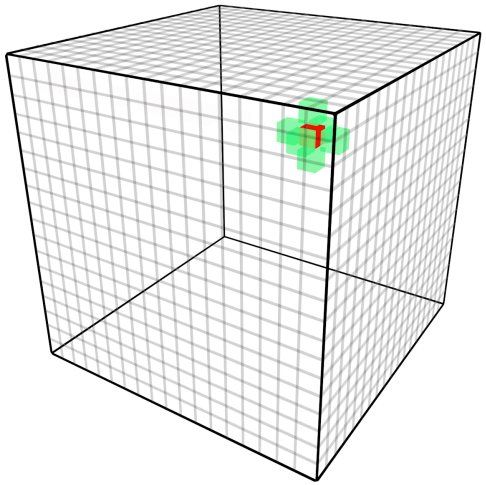


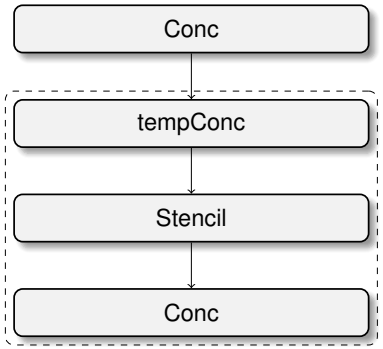


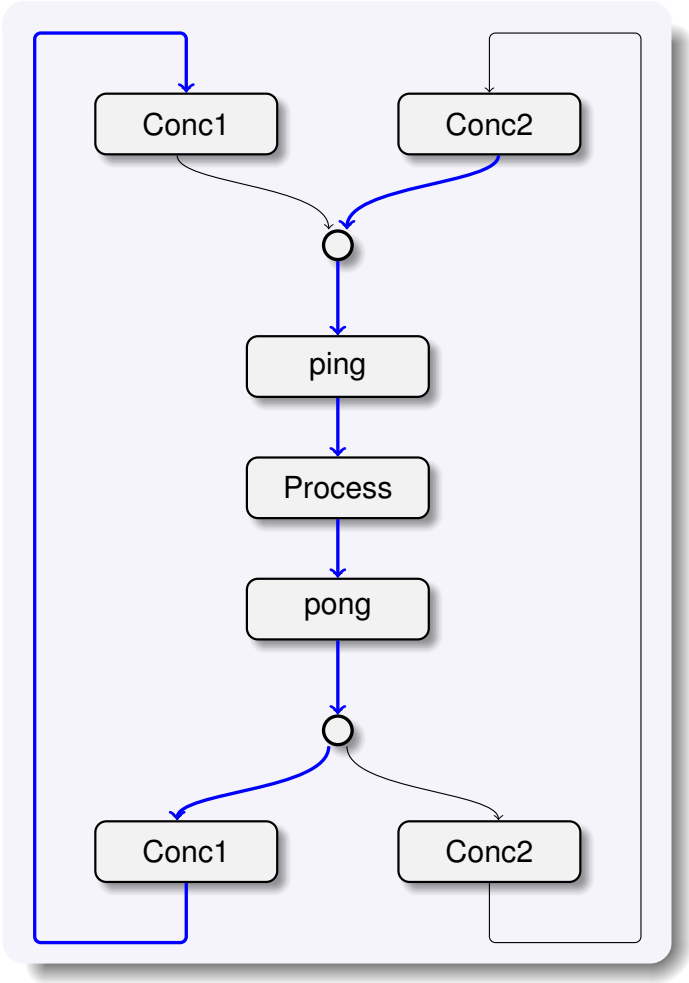


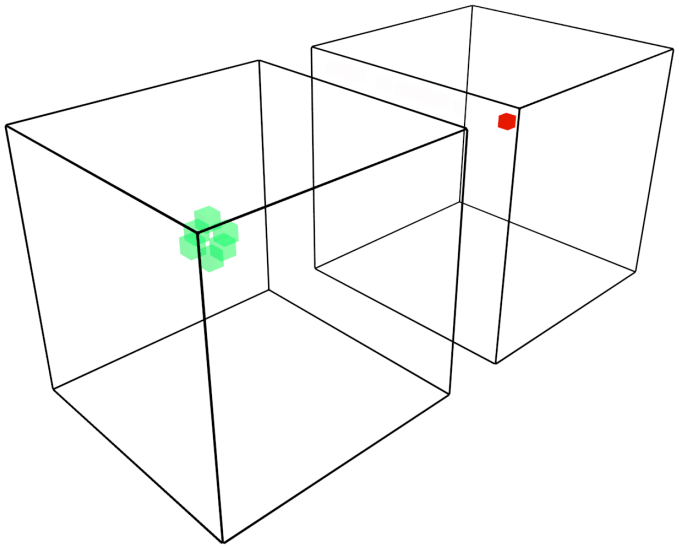


Optimization of runDiffusionStep









Production, diffusion and decay

runDecayStep

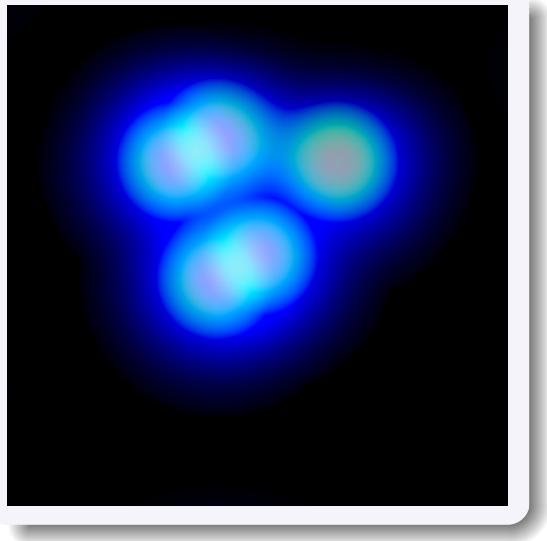
Computes the changes in substance concentrations due to decay

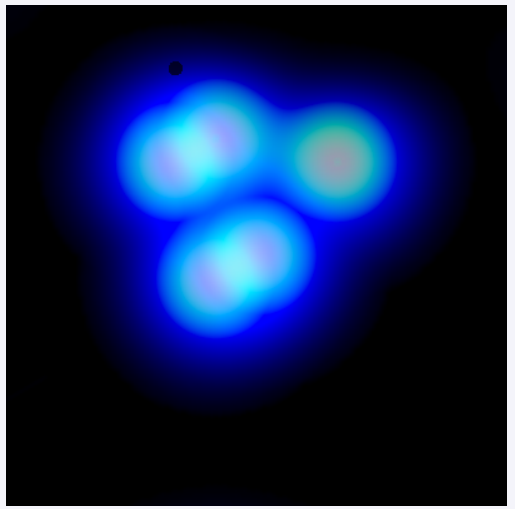
runDiffusionClusterStep

Computes movements of all cells based on gradients of the two substances

produceSubstances

Increases the concentration of substances at the location of the cells





Exploiting commonalities in Criterion computation

```
static float getEnergy(float** posAll, int* typesAll, int n, float spatialRange, int targetN) {
    getEnergy_sw.reset();
    // Computes an energy measure of clusteredness within a subvolume. The size of the subvolume
    // is computed by assuming roughly uniform distribution within the whole volume, and selecting
    // a volume comprising approximately targetN cells.
    int i1, i2;
    float currDist;

    float** posSubvol=0;    // array of all 3 dimensional cell positions
    posSubvol = new float*[n];
    int typesSubvol[n];

    float subVolMax = pow(float(targetN)/float(n),1.0/3.0)/2;

    if(quiet < 1)
        printf("subVolMax: %f\n", subVolMax);

    int nrCellsSubVol = 0;

    float intraClusterEnergy = 0.0;
    float extraClusterEnergy = 0.0;
    float nrSmallDist=0.0;

    for (i1 = 0; i1 < n; i1++) {
        posSubvol[i1] = new float[3];
        if ((fabs(posAll[i1][0]-0.5)<subVolMax) && (fabs(posAll[i1][1]-0.5)<subVolMax) && (fabs(posAll[i1][2]-0.5)<subVolMax)) {
            posSubvol[nrCellsSubVol][0] = posAll[i1][0];
            posSubvol[nrCellsSubVol][1] = posAll[i1][1];
            posSubvol[nrCellsSubVol][2] = posAll[i1][2];
            typesSubvol[nrCellsSubVol] = typesAll[i1];
            nrCellsSubVol++;
        }
    }

    for (i1 = 0; i1 < nrCellsSubVol; i1++) {
        for (i2 = i1+1; i2 < nrCellsSubVol; i2++) {
            currDist =  getL2Distance(posSubvol[i1][0],posSubvol[i1][1],posSubvol[i1][2],posSubvol[i2][0],posSubvol[i2][1],posSubvol[i2][2]);
            if (currDist<spatialRange) {
                nrSmallDist = nrSmallDist+1;//currDist/spatialRange;
                if (typesSubvol[i1]*typesSubvol[i2]>0) {
                    intraClusterEnergy = intraClusterEnergy+fmin(100.0,spatialRange/currDist); }
                else {
                    extraClusterEnergy = extraClusterEnergy+fmin(100.0,spatialRange/currDist);
                }
            }
        }
    }
    float totalEnergy = (extraClusterEnergy-intraClusterEnergy)/(1.0+100.0*nrSmallDist);
    getEnergy_sw.mark();
    return totalEnergy;
}
```

```
static bool getCriterion(float** posAll, int* typesAll, int n, float spatialRange, int targetN) {
    getCriterion_sw.reset();
    // Returns 0 if the cell locations within a subvolume of the total system, comprising approximately targetN cells,
    // are arranged as clusters, and 1 otherwise.

    int i1, i2;
    int nrClose=0;      // number of cells that are close (i.e. within a distance of spatialRange)
    float currDist;
    int sameTypeClose=0; // number of cells of the same type, and that are close (i.e. within a distance of spatialRange)
    int diffTypeClose=0; // number of cells of opposite types, and that are close (i.e. within a distance of spatialRange)

    float** posSubvol=0;    // array of all 3 dimensional cell positions in the subcube
    posSubvol = new float*[n];
    int typesSubvol[n];

    float subVolMax = pow(float(targetN)/float(n),1.0/3.0)/2;

    int nrCellsSubVol = 0;

    // the locations of all cells within the subvolume are copied to array posSubvol
    for (i1 = 0; i1 < n; i1++) {
        posSubvol[i1] = new float[3];
        if ((fabs(posAll[i1][0]-0.5)<subVolMax) && (fabs(posAll[i1][1]-0.5)<subVolMax) && (fabs(posAll[i1][2]-0.5)<subVolMax)) {
            posSubvol[nrCellsSubVol][0] = posAll[i1][0];
            posSubvol[nrCellsSubVol][1] = posAll[i1][1];
            posSubvol[nrCellsSubVol][2] = posAll[i1][2];
            typesSubvol[nrCellsSubVol] = typesAll[i1];

            nrCellsSubVol++;
        }
    }

    if(quiet < 1)
        printf("number of cells in subvolume: %d\n", nrCellsSubVol);


    // If there are not enough cells within the subvolume, the correctness criterion is not fulfilled
    if ((((float)(nrCellsSubVol))/(float)targetN) < 0.25) {
        getCriterion_sw.mark();
        if(quiet < 2)
            printf("not enough cells in subvolume: %d\n", nrCellsSubVol);
        return false;
    }

    // If there are too many cells within the subvolume, the correctness criterion is not fulfilled
    if ((((float)(nrCellsSubVol))/(float)targetN) > 4) {
        getCriterion_sw.mark();
        if(quiet < 2)
            printf("too many cells in subvolume: %d\n", nrCellsSubVol);
        return false;
    }
```

Vectorization of concentration volume

Memory alignment

- `_mm_malloc` and `_mm_free` to allocate and free aligned blocks of memory
- Lazy initialization

Linear access to 2d and 3d matrices with padding and alignment

- `Conc_Pad = L+(L%16 == 0? 0:16-L%16);` If the length is aligned to 16, do not modify it. Otherwise add the remaining size to be a multiple of 16.
- `float* Conc1 = (float*) _mm_malloc(2*Conc_Pad*Conc_Pad*Conc_Pad*sizeof(float), 64);` Align to 64 boundary
- `Conc[a][b][c][d]` ⇔
  `Conc[ a*Conc_Pad*Conc_Pad*Conc_Pad + b*Conc_Pad*Conc_Pad + c*Conc_Pad + d ]`

map-reduce

```
for (i1 = 0; i1 < nrCellsSubVol; i1++) {
    for (i2 = i1+1; i2 < nrCellsSubVol; i2++) {
        currDist =  getL2Distance(posSubvol[i1][0] ...
        if (currDist<spatialRange) {
            nrSmallDist = nrSmallDist+1;
            if (typesSubvol[i1]*typesSubvol[i2]>0) {
                intraClusterEnergy += fmin(100.0,spatialRange/currDist);
             } else {
                extraClusterEnergy += fmin(100.0,spatialRange/currDist);
            }
        }
    }
}
```

expensive computation

loop dependency

```
for (i1 = 0; i1 < nrCellsSubVol; i1++) {
    for (i2 = i1+1; i2 < nrCellsSubVol; i2++) {
        currDist =  getL2Distance(posSubvol[i1][0] ...
        if (currDist<spatialRange) {
            nrClose++;
            if (typesSubvol[i1]*typesSubvol[i2]<0) {
                diffTypeClose++;
            } else {
                sameTypeClose++;
            }
        }
    }
}
```

expensive computation

loop dependency

map-reduce

```
for (i1 = 0; i1 < nrCellsSubVol; i1++) {
    for (i2 = i1+1; i2 < nrCellsSubVol; i2++) {
        currDist =  getL2Distance(posSubvol[i1][0] ...
```

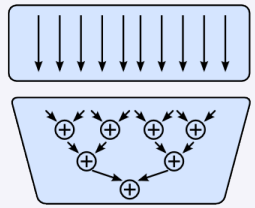
nrSmallDist_matrix

intraClusterEnergy_matrix

extraClusterEnergy_matrix

$$\frac{1}{\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}}$$

$$\frac{\text{nrCellsSubVol} * (\text{nrCellsSubVol} - 1)}{2 \cdot \text{nr\_cores}}$$



map

reduce

$$\log\left(\frac{\text{nrCellsSubVol} * (\text{nrCellsSubVol} - 1)}{2 \cdot \text{nr\_cores}}\right)$$

```
#pragma omp parallel for parallel reduction ( + : nrSmallDist, intra
for (int i1 = 0; i1 < nrCellsSubVol; i1++) {
    for (int i2 = i1+1; i2 < nrCellsSubVol; i2++) {
            nrSmallDist += nrSmallDist_matrix(i1, i2);
            intraClusterEnergy += intraClusterEnergy_matrix(i1,i2);
            extraClusterEnergy += extraClusterEnergy_matrix(i1,i2);
```

Vectorization of random number generation

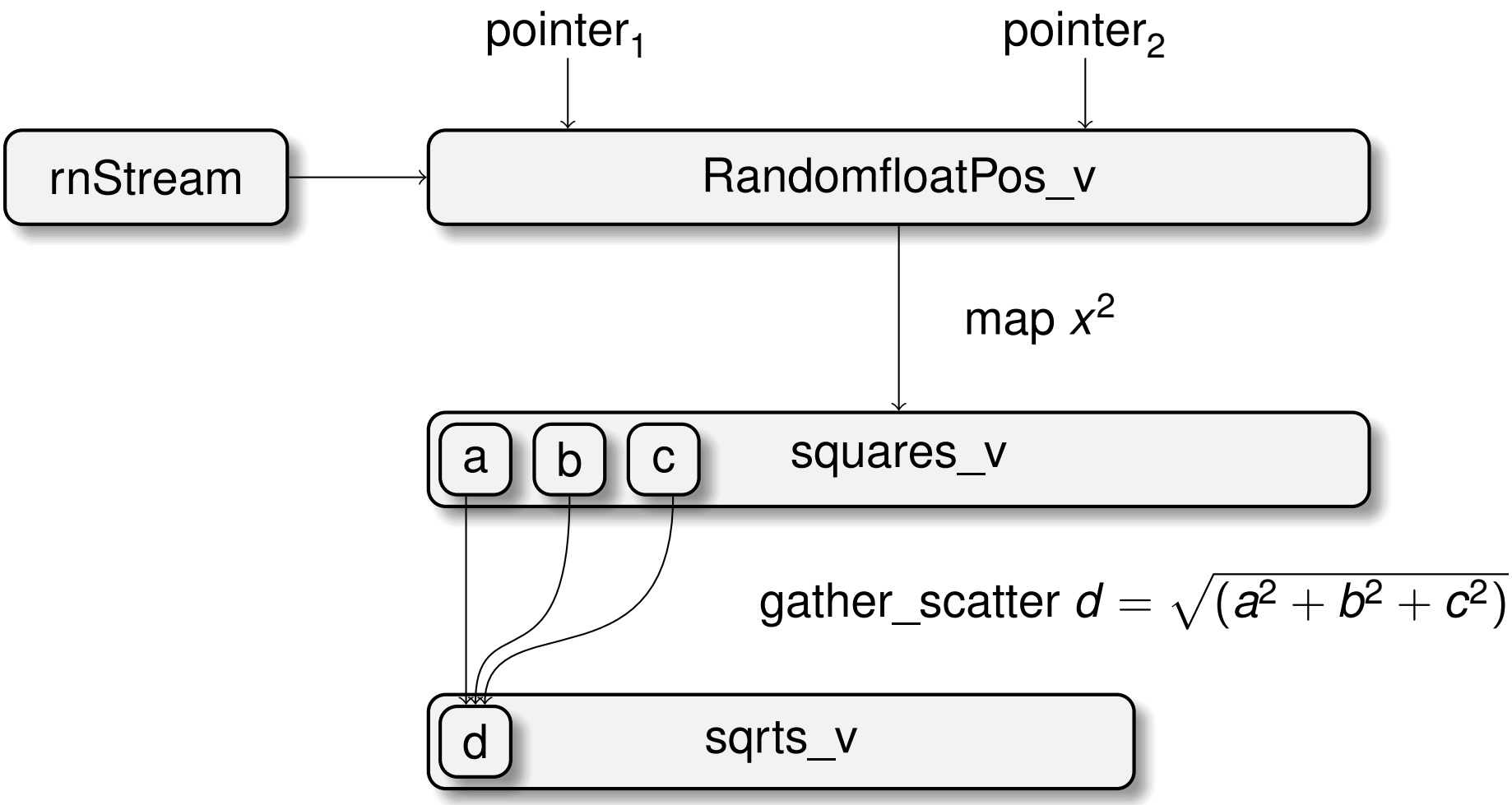
Intel vectorized random number generator

- Create and initialize a random number generator
- Create an array for receiving the random numbers
- Fill the array

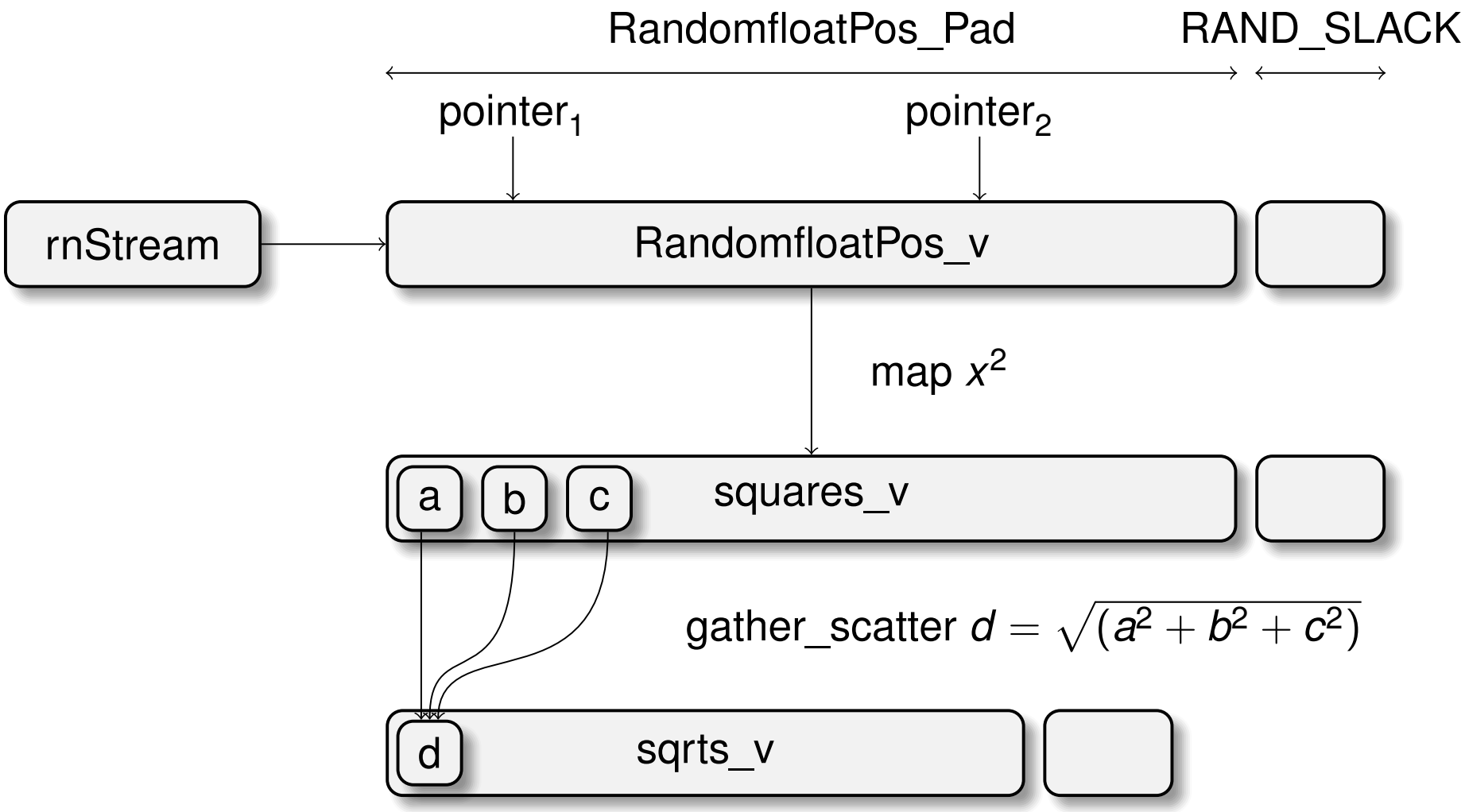
```
VSLStreamStatePtr rnStream;
vslNewStream( &rnStream, VSL_BRNG_R250, 0 );
int RandomFloatPos_Pad   = (3*size)+((3*size)%16 == 0?0:16-(3*size)%16);
RandomFloatPos_v = (float*) _mm_malloc(RandomFloatPos_Pad*sizeof(float), 16);
vsRngUniform( VSL_RNG_METHOD_UNIFORM_STD, rnStream, RandomFloatPos_Pad, RandomFloatPos_v, -0.5f, 0.5f);
```

https://software.intel.com/en-us/mkl-developer-reference-c-vslnewstream

Vectorization of random number generation



Vectorization of random number generation



Loop fusion

Cells out of the simulation domain

After diffusion, cluster and decay, the cells that are outside the simulation domain are 'pushed' back into it

```
produceSubstances(Conc, posAll, typesAll, L, n);
runDiffusionStep(Conc, Conc2, L, D);
runDecayStep(Conc2, L, mu);
runDiffusionClusterStep(Conc2, currMov, posAll, typesAll, n, L, speed);
std::swap(Conc,Conc2);

for (c=0; c<n; c++) {
    posAll[c][0] = posAll[c][0]+currMov[c][0];
    posAll[c][1] = posAll[c][1]+currMov[c][1];
    posAll[c][2] = posAll[c][2]+currMov[c][2];

    // boundary conditions: cells can not move out of the cube [0,1]^3
    for (d=0; d<3; d++) {
        if (posAll[c][d]<0) {posAll[c][d]=0;}
        if (posAll[c][d]>1) {posAll[c][d]=1;}
    }
}
```

Loops over c

Loop fusion

Cells out of the simulation domain

After diffusion, cluster and decay, the cells that are outside the simulation domain are 'pushed' back into it

```
produceSubstances(Conc, posAll, typesAll, L, n);
runDiffusionStep(Conc, Conc2, L, D);
runDecayStep(Conc2, L, mu);
runDiffusionClusterStep(Conc2, currMov, posAll, typesAll, n, L, speed);
std::swap(Conc,Conc2);

for (c=0; c<n; c++) {
    posAll[c][0] = posAll[c][0]+currMov[c][0];
    posAll[c][1] = posAll[c][1]+currMov[c][1];
    posAll[c][2] = posAll[c][2]+currMov[c][2];

    // boundary conditions: cells can not move out of the cube [0,1]^3
    for (d=0; d<3; d++) {
        if (posAll[c][d]<0) {posAll[c][d]=0;}
        if (posAll[c][d]>1) {posAll[c][d]=1;}
    }
}
```

} Also loops over c

} Loops over c

Loop reordering

Access pattern does not match memory structure

```
#pragma omp parallel for collapse(2)
    for (int i1 = 0; i1 < L; i1++) {
        for (int i2 = 0; i2 < L; i2++) {
#pragma simd
            for (int i3 = 0; i3 < L; i3++) {
                int xUp = (i1+1);
                int xDown = (i1-1);
                int yUp = (i2+1);
                int yDown = (i2-1);
                int zUp = (i3+1);
                int zDown = (i3-1);

                for (int subInd = 0; subInd < 2; subInd++) {
                        pong(subInd,i1,i2,i3) = ping(subInd,i1,i2,i3);
                    if (xUp<L) {
                        pong(subInd,i1,i2,i3) += (ping(subInd,xUp,i2,i3)-ping(subInd,i1,i2,i3))*D/6;
```

- In memory: subInd, x, y, z
- Access: x, y, z, subInd

Loop fusion

Cache misses

Why the reordering of loops in runDiffusionStep affects the execution time of runDecayStep?

```
#pragma omp parallel for collapse(3)
    for (int subInd = 0; subInd < 2; subInd++) {
    for (int i1 = 0; i1 < L; i1++) {
        for (int i2 = 0; i2 < L; i2++) {
#pragma simd
            for (int i3 = 0; i3 < L; i3++) {
                int xUp = (i1+1);
                int xDown = (i1-1);
                int yUp = (i2+1);
                int yDown = (i2-1);
                int zUp = (i3+1);
                int zDown = (i3-1);

                    pong(subInd,i1,i2,i3) = ping(subInd,i1,i2,i3);
                if (xUp<L) {
                    pong(subInd,i1,i2,i3) += (ping(subInd,xUp,i2,i3)
                }
```

```
static void runDecayStep(float* Conc, int L, float mu) {
    runDecayStep_sw.reset();
    // computes the changes in substance concentrations due to decay
#pragma omp parallel for collapse(2)
    for (int i1 = 0; i1 < L; i1++) {
        for (int i2 = 0; i2 < L; i2++) {
#pragma simd
            for (int i3 = 0; i3 < L; i3++) {
                Conc(0,i1,i2,i3) = Conc(0,i1,i2,i3)*(1-mu);
                Conc(1,i1,i2,i3) = Conc(1,i1,i2,i3)*(1-mu);
            }
        }
    }
    runDecayStep_sw.mark();
}
```

Loops can be fused

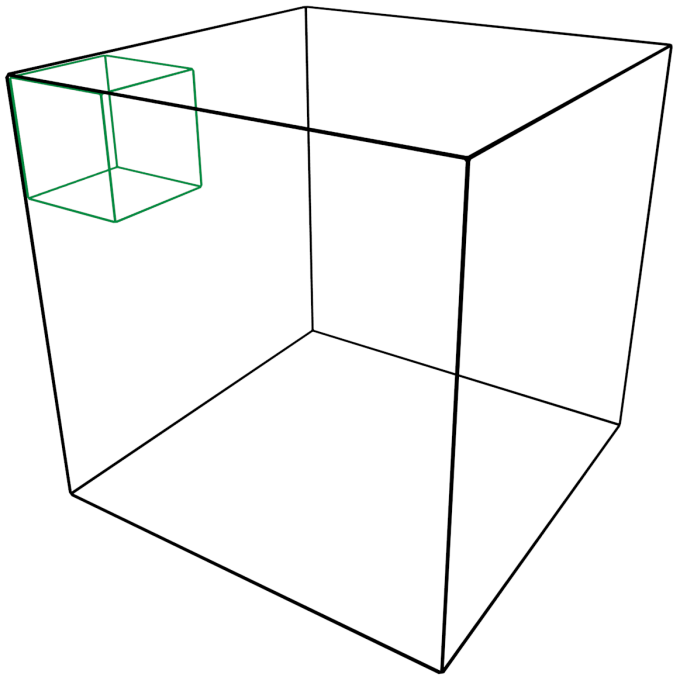
Loop tiling and cache-oblivious implementation

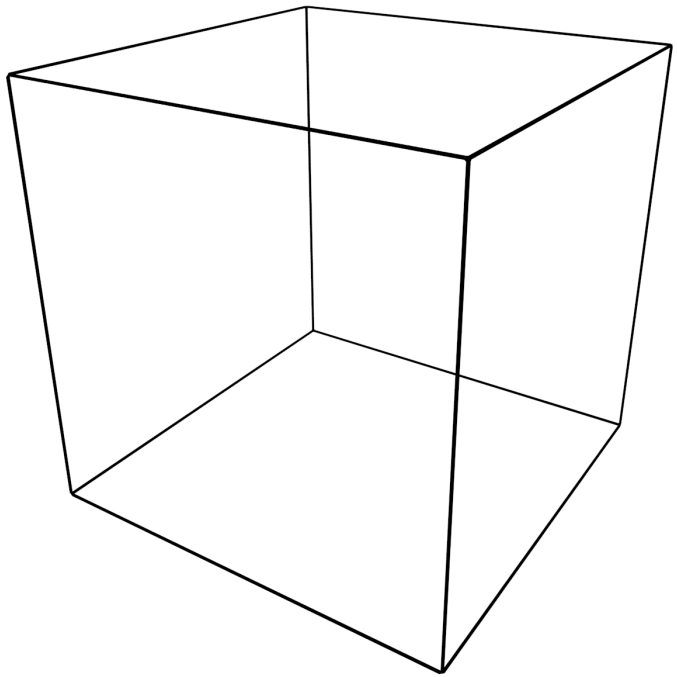
Loop-tiling

2 levels of iterations: tile, voxel

Cache-Oblivious

Recurse until the base-case fits in cache





Extra considerations

- restrict : helps compiler to reason about pointer aliasing
- pragma vector aligned: promises that pointers always read or write memory with an aligned access
- static functions: are only visible to other functions in the same file
- position and movement can also be vectorized

Compiler, Broadwell and KNL

Compiler flags

- -O3 : We forgot this one ;-)
- -fp-model fast=2 : MKL
- -fimf-precision=low : MKL

Broadwell

- -xCORE-AVX2 : Use generic AVX instructions
- -par-affinity=compact : assign threads to particular CPUs in the platform

Knights Landing

- -xMIC-AVX512 : Use AVX512 instructions
- -par-affinity=balanced : Separate threads until all cores have at least one thread
- -par-num-threads=136 : Use a maximim of 136 threads
- -l nodes=1:knl7250:flat; numactl -m 1 ./cell_clustering : put the entire program in the high-bandwidth memory

Final results

Small

```
speed           = 0.01
T               = 500
L               = 80
D               = 0.3
mu              = 0.1
divThreshold    = 16
pathThreshold   = 2.0
spatialScale    = 5.0
```

~ 1 sq centimeter of tissue

Huge

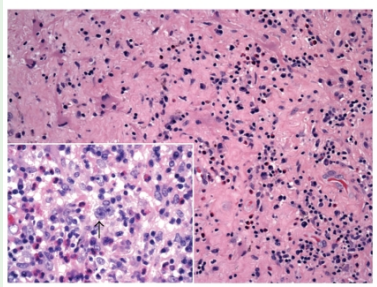
```
speed           = 0.01
T               = 500
L               = 820
D               = 0.3
mu              = 0.1
divThreshold    = 25
pathThreshold   = 2.0
spatialScale    = 5.0
```

~ a Brodmann's area
such as the visual cortex

Final results

Small

```
speed                = 0.01
T
L
D
mu
divTh
pathT
spati
```



~ 1 sq centimeter of tissue

Huge

```
speed                = 0.01
T                    = 500
L                    = 820
D                    = 0.3
mu                   = 0.1
divThreshold         = 25
pathThreshold        = 2.0
spatialScale         = 5.0
```

~ a Brodmann's area
such as the visual cortex

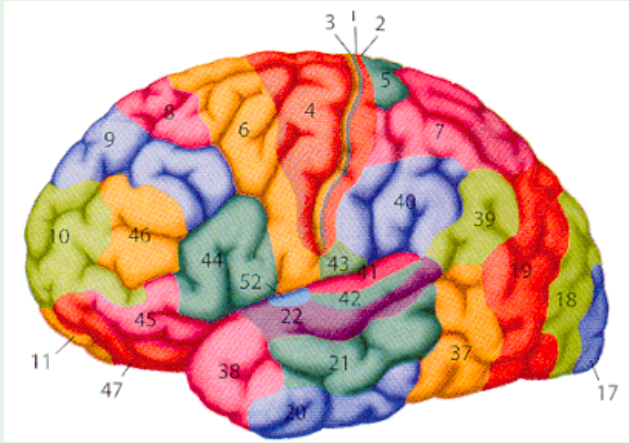
Final results

Small

```
speed          = 0.01
T              = 500
L              = 80
D              = 0.3
mu             = 0.1
divThreshold   = 16
pathThreshold  = 2.0
spatialScale   = 5.0
```

~ 1 sq centimeter of tissue

Huge



such as the visual cortex

Final results

Small

```
speed           = 0.01
T               = 500
L               = 80
D               = 0.3
mu              = 0.1
divThreshold    = 16
pathThreshold   = 2.0
spatialScale    = 5.0
```

~ 1 sq centimeter of tissue

Huge

```
speed           = 0.01
T               = 500
L               = 820
D               = 0.3
mu              = 0.1
divThreshold    = 25
pathThreshold   = 2.0
spatialScale    = 5.0
```

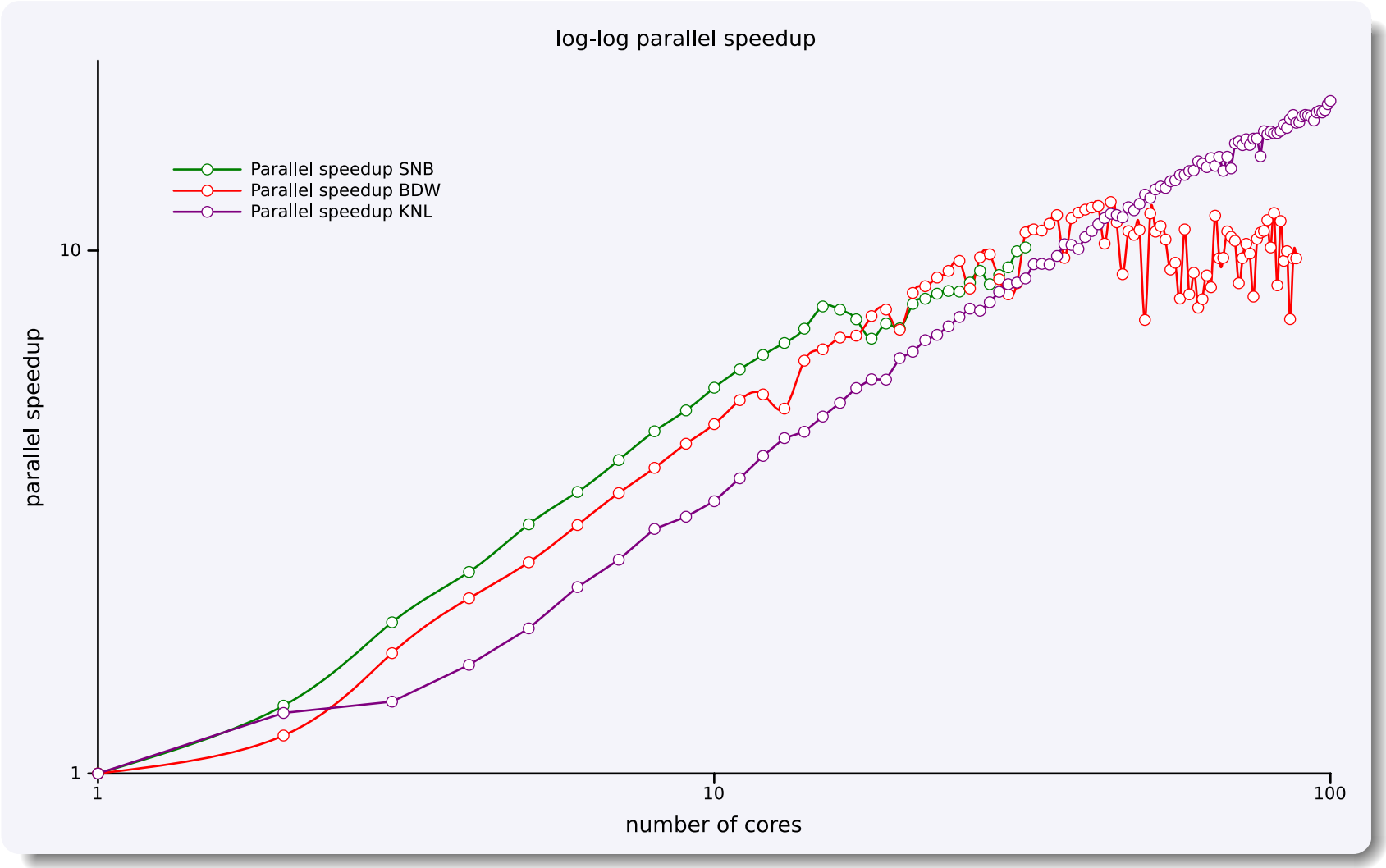
~ a Brodmann's area
such as the visual cortex

Time before and after optimization (lower is better).

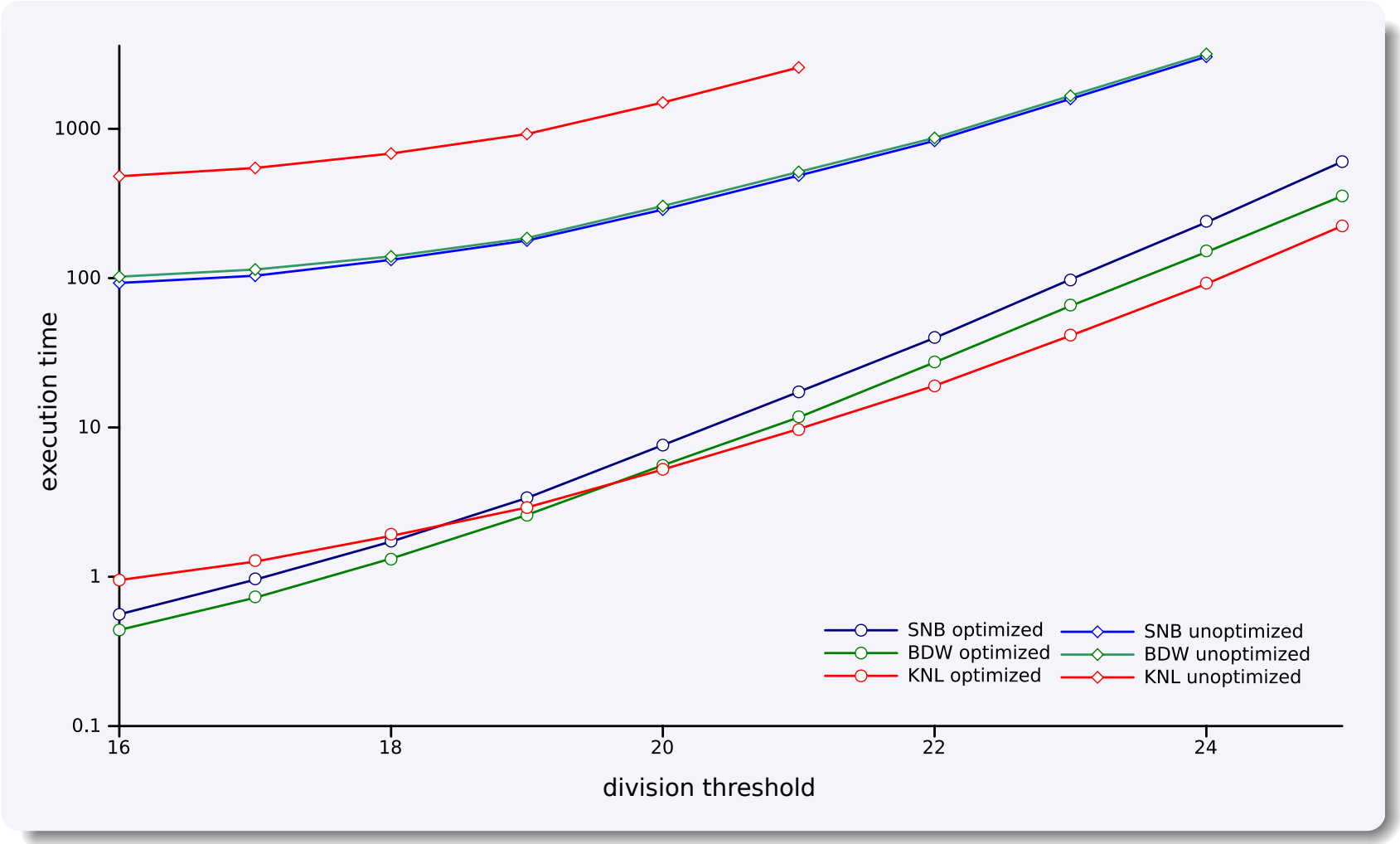
	Before	After	speedup
SNB	92.82 s	0.56 s	165x
BDW	102.21 s	0.44 s	232x
KNL	481.15 s	0.95 s	506x

	Before	After
SNB	n/a	602.87 s
BDW	n/a	355.03 s
KNL	n/a	223.71 s

Parallel speedup



Scalability



Overall Conclusions

Conclusions

- Thanks for coming
- Some techniques to improve execution time in HPC architectures
- It is important to understand what we're doing both in terms of choosing a platform and also in optimizing the code for that platform

pablo.aledo@gmail.com