An optimization approach for the computational modeling of biological development

Pablo González de Aledo

July 2017

MC² Series
Agent-based Simulations

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

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

Competition winners Daniel Wei (second from left) and Pablo Gonzalez De Aledo Munigan (second from right), with their partners and Russell Beutler of Intel (far left). (Image: Andrew Purcell/CERN)
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
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 MacQuarrie 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

Considerations

- 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
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
Replicating our results

ssh colfax
make
make run-knl
make run-snb
make run-bdw
2 Phases
- Initial cells creation
- Cell clustering
# Moving pieces

<table>
<thead>
<tr>
<th>cells</th>
<th>substances</th>
</tr>
</thead>
<tbody>
<tr>
<td>- move</td>
<td>- increase/decrease</td>
</tr>
<tr>
<td>- know how much they have travelled</td>
<td>- move to neighbouring voxels</td>
</tr>
<tr>
<td>- ‘sense’ chemicals</td>
<td></td>
</tr>
<tr>
<td>- divide</td>
<td></td>
</tr>
<tr>
<td>- cluster</td>
<td></td>
</tr>
<tr>
<td>- produce two kind of substances</td>
<td></td>
</tr>
</tbody>
</table>

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

- Instrumentation
- Identification of Parallel Patterns
- Optimization

Obstacles to parallelization

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

```java
static void runDiffusionStep(float**** Conc, int L, float D) {
    runDiffusionStep_sw.reset();
    // computes the changes in substance concentrations due to diffusion
    int l1, l2, l3, subInd;
    float tempConc[l1][l2][13][13];
    for (l1 = 0; l1 < L; l1++) {
        for (l2 = 0; l2 < L; l2++) {
            for (l3 = 0; l3 < L; l3++) {
                tempConc[l1][l2][l3] = Conc[l1][l2][l3][l3];
            }
        }
    }

    int xUp, xDown, yUp, yDown, zUp, zDown;
    for (l1 = 0; l1 < L; l1++) {
        for (l2 = 0; l2 < L; l2++) {
            for (l3 = 0; l3 < L; l3++) {
                xUp = (l1+1);
                xDown = (l1-1);
                yUp = (l2+1);
                yDown = (l2-1);
                zUp = (l3+1);
                zDown = (l3-1);
                for (subInd = 0; subInd < 2; subInd++) {
                    if (xUp < L) {
                        Conc[subInd][l1][l2][l3] += (tempConc[subInd][xUp][l2][l3]-tempConc[subInd][l1][l2][l3])*D/6;
                    }
                    if (xDown >= 0) {
                        Conc[subInd][l1][l2][l3] += (tempConc[subInd][xDown][l2][l3]-tempConc[subInd][l1][l2][l3])*D/6;
                    }
                    if (yUp < L) {
                        Conc[subInd][l1][l2][l3] += (tempConc[subInd][l1][yUp][l3]-tempConc[subInd][l1][l2][l3])*D/6;
                    }
                    if (yDown >= 0) {
                        Conc[subInd][l1][l2][l3] += (tempConc[subInd][l1][yDown][l3]-tempConc[subInd][l1][l2][l3])*D/6;
                    }
                    if (zUp < L) {
                        Conc[subInd][l1][l2][l3] += (tempConc[subInd][l1][l2][zUp]-tempConc[subInd][l1][l2][l3])*D/6;
                    }
                    if (zDown >= 0) {
                        Conc[subInd][l1][l2][l3] += (tempConc[subInd][l1][l2][zDown]-tempConc[subInd][l1][l2][l3])*D/6;
                    }
                }
            }
        }
    }
    runDiffusionStep_sw.mark();
}
```
Optimization of runDiffusionStep
Optimization of runDiffusionStep

Conc

tempConc

Stencil

Conc

Map

Conc1

ping

Process

pong

Conc1

Conc2
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_wv(res); // Compute an energy measure of clusters within a subvolume. The sum of the subvolume
                        // is computed by summing up the energy of the whole volume, and selecting
                        // a volume comprising approximately targetN cells.
    int i, j, k;
    float curdist;
    float* posSubVol; // 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); //
    if (quiet < 1)
        printf("subVolMax = %0.4f
", subVolMax);
    int nrCellsSubVol = 0;
    float intraClusterEnergy = 0.0;
    float extraClusterEnergy = 0.0;
    float nrmSmallDist = 0.0;
    for (i = 0; i < n; i++) {
        posSubVol[i] = posAll[i];
        if (fabs(posAll[i][0]-posSubVol[i][0]) < subVolMax && fabs(posAll[i][1]-posSubVol[i][1]) < subVolMax &&
            fabs(posAll[i][2]-posSubVol[i][2]) < subVolMax) {
            posSubVol[nrCellsSubVol] = posAll[i];
            nrCellsSubVol++;
            nrmSmallDist = sqrt(pow(posSubVol[i][0]-posAll[0], 2) +
                                pow(posSubVol[i][1]-posAll[1], 2) +
                                pow(posSubVol[i][2]-posAll[2], 2));
        }
    }
    for (i = 0; i < n; i++) {
        for (j = i+1; j < n; j++) {
            curdist = getDistance(posSubVol[i], posSubVol[j], posSubVol[i], posSubVol[j], posSubVol[j], posSubVol[j], posSubVol[j], posSubVol[j], posSubVol[j]);
            if (curdist < spatialRange) {
                nrmSmallDist = sqrt(pow(posSubVol[i][0]-posSubVol[j][0], 2) +
                                    pow(posSubVol[i][1]-posSubVol[j][1], 2) +
                                    pow(posSubVol[i][2]-posSubVol[j][2], 2));
                if (typesSubVol[i] == typesSubVol[j]) { // typesSubVol[i] = typesSubVol[j]
                    extraClusterEnergy += intraClusterEnergy * fabs(100.0, spatialRange, curdist);
                }
            }
        }
    }
    float totalEnergy = (extraClusterEnergy * intraClusterEnergy)/subVolMax;
    return totalEnergy;
}
```

```
static bool getCriterion(float* posAll, int* typesAll, int n, float spatialRange, int targetN) {
    getCriterion_wv(res); // Return if the cell locations within a subvolume of the total system, comprising approximately targetN cells,
                          // are arranged as clusters, and 1 otherwise.
    int i, j, k;
    int nrmClosed; // number of cells that are close (i.e. within a distance of spatialRange)
    float curdist;
    int sameTypeClosed; // number of cells of the same type, and that are close (i.e. within a distance of spatialRange)
    int diffTypeClosed; // number of cells of opposite types, and that are close (i.e. within a distance of spatialRange)
    float* posSubVol; // 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); //
    int nrCellsSubVol = 0;
    // the locations of all cells within the subvolume are copied to array posSubVol
    for (i = 0; i < n; i++)
        posSubVol[i] = posAll[i];
    if (quiet < 1)
        printf("nrCellsSubVol = %0.4f
", nrCellsSubVol);
    if (curdist < spatialRange)
        return false;
    if (nrCellsSubVol < subVolMax) {
        return false;
    }
    if (nrCellsSubVol > subVolMax) {
        return false;
    }
    for (i = 0; i < n; i++) {
        if (quiet < 1)
            print("number of cells in subvolume: %0.4f, nrCellsSubVol=%0.4f
", nrCellsSubVol);
        if (nrCellsSubVol < subVolMax) {
            return false;
        }
        if (nrCellsSubVol > subVolMax) {
            return false;
        }
        if (diffTypeClosed < 0.75) {
            getCriterion_wv(res);
            if (quiet < 1)
                print("Too many cells in subvolume: %0.4f, nrCellsSubVol=%0.4f
", nrCellsSubVol);
            return false;
        }
    }
    return true;
}
```
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]`
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);
            }
        }
    }
}

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++;
            }
        }
    }
}
map-reduce

```c
for (i1 = 0; i1 < nrCellsSubVol; i1++) {
    for (i2 = i1+1; i2 < nrCellsSubVol; i2++) {
        currDist = getL2Distance(posSubvol[i1][0] ...)
        nrSmallDist_matrix
        intraClusterEnergy_matrix
        extraClusterEnergy_matrix
        nrCellsSubVol * (nrCellsSubVol - 1)
        2 * nr_cores
        1
        \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}

#pragma omp parallel for 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

```c
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);
```

Vectorization of random number generation

\[
gather\_scatter \ d = \sqrt{a^2 + b^2 + c^2}
\]
Vectorization of random number generation

Pablo González de Aledo
MC2 Series
July 2017
Cells out of the simulation domain

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

```cpp
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
Cells out of the simulation domain

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

```cpp
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;}
}
```
Loop reordering

Access pattern does not match memory structure

```c
#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<i1) {
                            pong(subInd,i1,i2,i3) += (ping(subInd,xUp,i2,i3)-ping(subInd,i1,i2,i3))*D/6;
                        }
                    }
                }
            }
        }
    }
```
Cache misses

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

Loops can be fused

```c
#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 omp 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,i2,13) = ping(subInd,i1,i2,13);
                if (xUp<L) {
                    pong(subInd,i1,i2,13) += (ping(subInd,xUp,i2,13)
                }
            }
        }
    }
}
```
## Loop tiling and cache-oblivious implementation

<table>
<thead>
<tr>
<th>Loop-tiling</th>
<th>Cache-Oblivious</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 levels of iterations: tile, voxel</td>
<td>Recurse until the base-case fits in cache</td>
</tr>
</tbody>
</table>
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 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 maximum of 136 threads  
- **-l nodes=1:knl7250:flat; numactl -m 1 ./cell_clustering**: put the entire program in the high-bandwidth memory
## Final results

<table>
<thead>
<tr>
<th>Small</th>
<th>Huge</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed = 0.01</td>
<td>speed = 0.01</td>
</tr>
<tr>
<td>T = 500</td>
<td>T = 500</td>
</tr>
<tr>
<td>L = 80</td>
<td>L = 820</td>
</tr>
<tr>
<td>D = 0.3</td>
<td>D = 0.3</td>
</tr>
<tr>
<td>mu = 0.1</td>
<td>mu = 0.1</td>
</tr>
<tr>
<td>divThreshold = 16</td>
<td>divThreshold = 25</td>
</tr>
<tr>
<td>pathThreshold = 2.0</td>
<td>pathThreshold = 2.0</td>
</tr>
<tr>
<td>spatialScale = 5.0</td>
<td>spatialScale = 5.0</td>
</tr>
</tbody>
</table>

~ 1 sq centimeter of tissue

~ a Brodmann's area such as the visual cortex
Final results

~ 1 sq centimeter of tissue

~ a Brodmann's area such as the visual cortex
## Final results

<table>
<thead>
<tr>
<th>Small</th>
<th>Huge</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>speed</strong> = 0.01</td>
<td><strong>speed</strong> = 0.01</td>
</tr>
<tr>
<td><strong>T</strong> = 500</td>
<td><strong>T</strong> = 500</td>
</tr>
<tr>
<td><strong>L</strong> = 80</td>
<td><strong>L</strong> = 820</td>
</tr>
<tr>
<td><strong>D</strong> = 0.3</td>
<td><strong>D</strong> = 0.3</td>
</tr>
<tr>
<td><strong>mu</strong> = 0.1</td>
<td><strong>mu</strong> = 0.1</td>
</tr>
<tr>
<td><strong>divThreshold</strong> = 16</td>
<td><strong>divThreshold</strong> = 25</td>
</tr>
<tr>
<td><strong>pathThreshold</strong> = 2.0</td>
<td><strong>pathThreshold</strong> = 2.0</td>
</tr>
<tr>
<td><strong>spatialScale</strong> = 5.0</td>
<td><strong>spatialScale</strong> = 5.0</td>
</tr>
</tbody>
</table>

~ 1 sq centimeter of tissue such as the visual cortex
## Final results

### Small

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed</td>
<td>0.01</td>
</tr>
<tr>
<td>T</td>
<td>500</td>
</tr>
<tr>
<td>L</td>
<td>80</td>
</tr>
<tr>
<td>D</td>
<td>0.3</td>
</tr>
<tr>
<td>mu</td>
<td>0.1</td>
</tr>
<tr>
<td>divThreshold</td>
<td>16</td>
</tr>
<tr>
<td>pathThreshold</td>
<td>2.0</td>
</tr>
<tr>
<td>spatialScale</td>
<td>5.0</td>
</tr>
</tbody>
</table>

~ 1 sq centimeter of tissue

### Huge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed</td>
<td>0.01</td>
</tr>
<tr>
<td>T</td>
<td>500</td>
</tr>
<tr>
<td>L</td>
<td>820</td>
</tr>
<tr>
<td>D</td>
<td>0.3</td>
</tr>
<tr>
<td>mu</td>
<td>0.1</td>
</tr>
<tr>
<td>divThreshold</td>
<td>25</td>
</tr>
<tr>
<td>pathThreshold</td>
<td>2.0</td>
</tr>
<tr>
<td>spatialScale</td>
<td>5.0</td>
</tr>
</tbody>
</table>

~ a Brodmann's area such as the visual cortex

### Time before and after optimization (lower is better).

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNB</td>
<td>92.82 s</td>
<td>0.56 s</td>
<td>165x</td>
</tr>
<tr>
<td>BDW</td>
<td>102.21 s</td>
<td>0.44 s</td>
<td>232x</td>
</tr>
<tr>
<td>KNL</td>
<td>481.15 s</td>
<td>0.95 s</td>
<td>506x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNB</td>
<td>n/a</td>
<td>602.87 s</td>
</tr>
<tr>
<td>BDW</td>
<td>n/a</td>
<td>355.03 s</td>
</tr>
<tr>
<td>KNL</td>
<td>n/a</td>
<td>223.71 s</td>
</tr>
</tbody>
</table>
Parallel speedup

log-log parallel speedup

- Parallel speedup SNB
- Parallel speedup BDW
- Parallel speedup KNL

parallel speedup vs. number of cores
Scalability

![Scalability Graph]

- SNB optimized
- SNB unoptimized
- BDW optimized
- BDW unoptimized
- KNL optimized
- KNL unoptimized

**Execution Time** vs **Division Threshold**
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