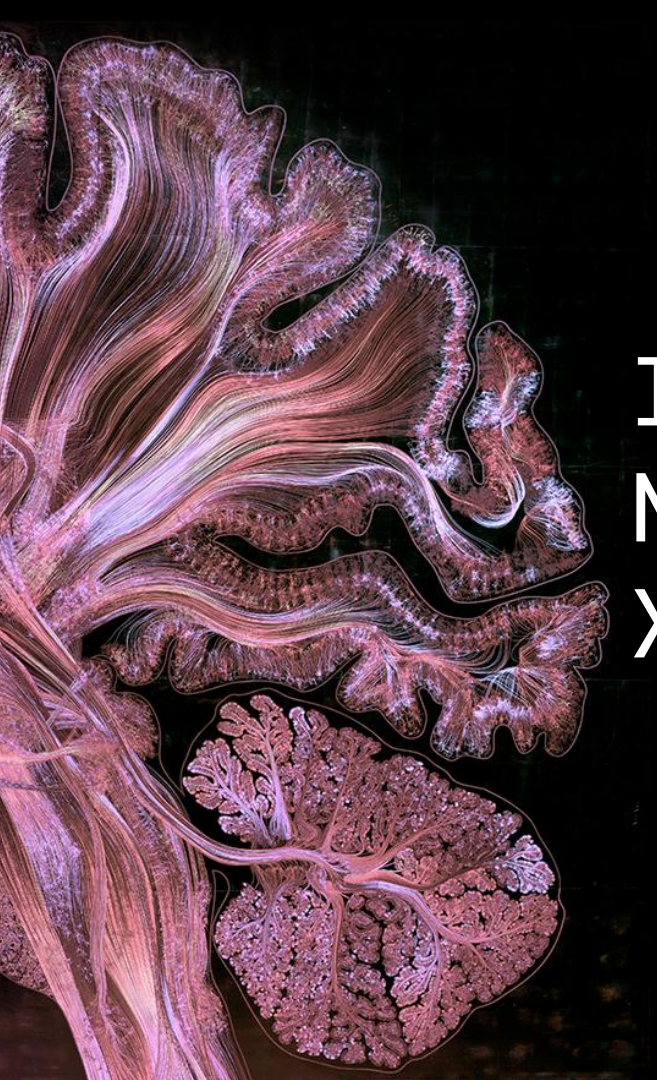
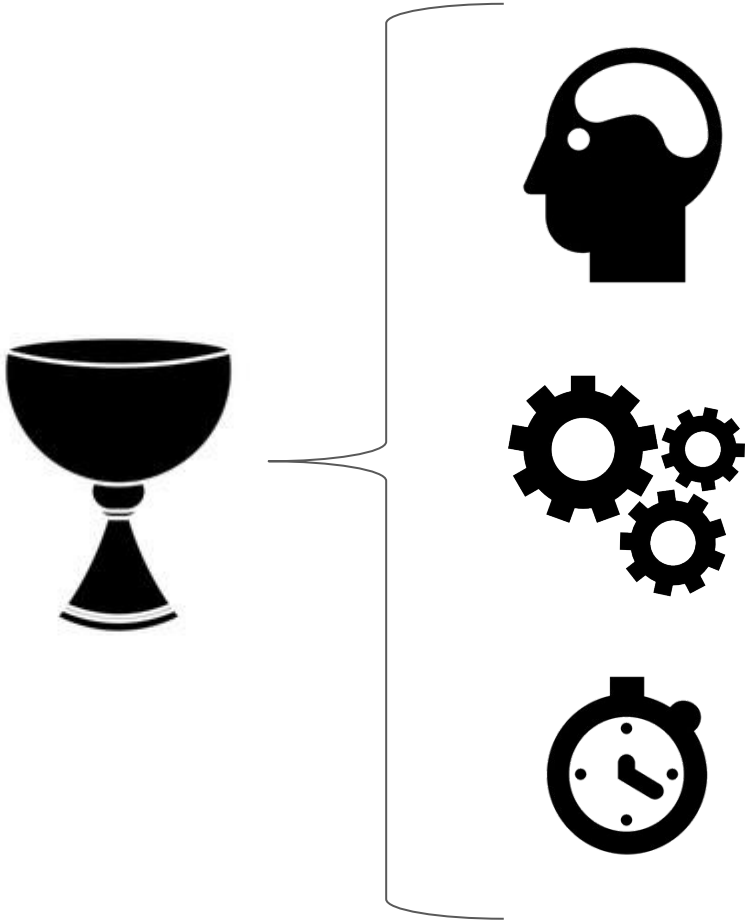


InfOli: A Parallel Neuron Simulator on Xeon Phi Architectures

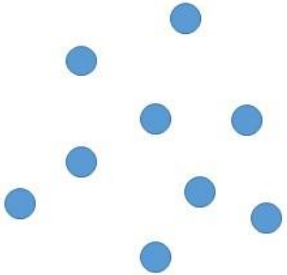
Diego Jiménez Vargas

Computational Neuroscience

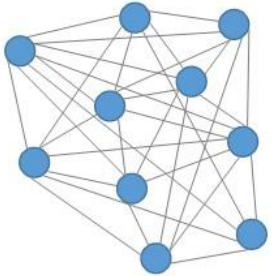


Problem Complexity

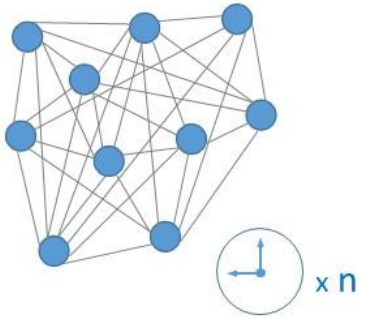
Many FLOPs per neuron



Massive network

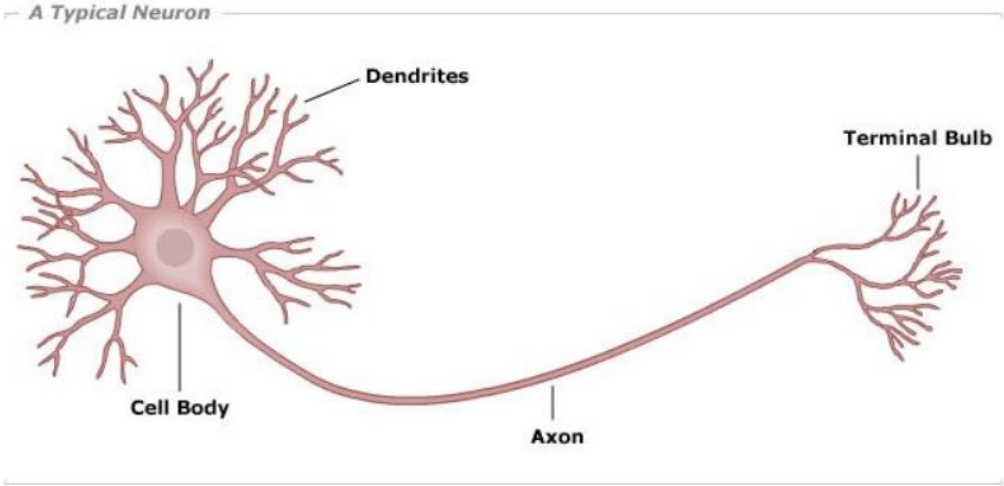


Densely connected networks



Real-time response is currently imposible

InfOli Simulator - Description



Tri-compartmental model

Soma (body): computation

Dendrite: communication

Axon: output

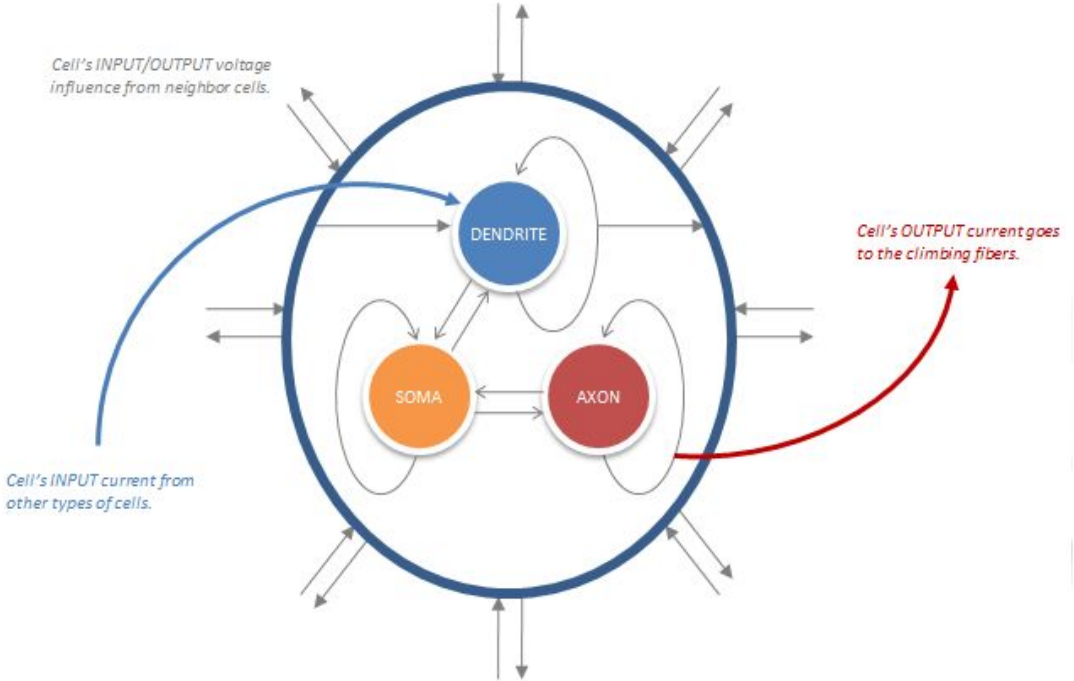
→

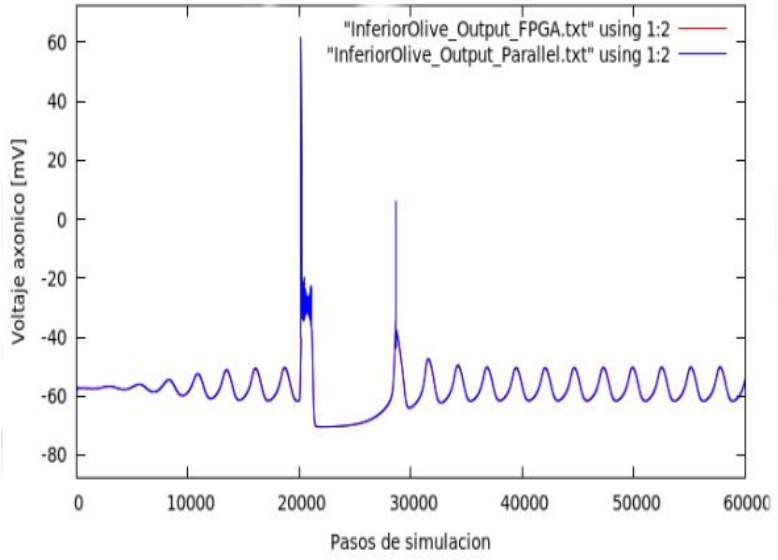
Gap Junction Mechanism

Communication between dendrites in the network

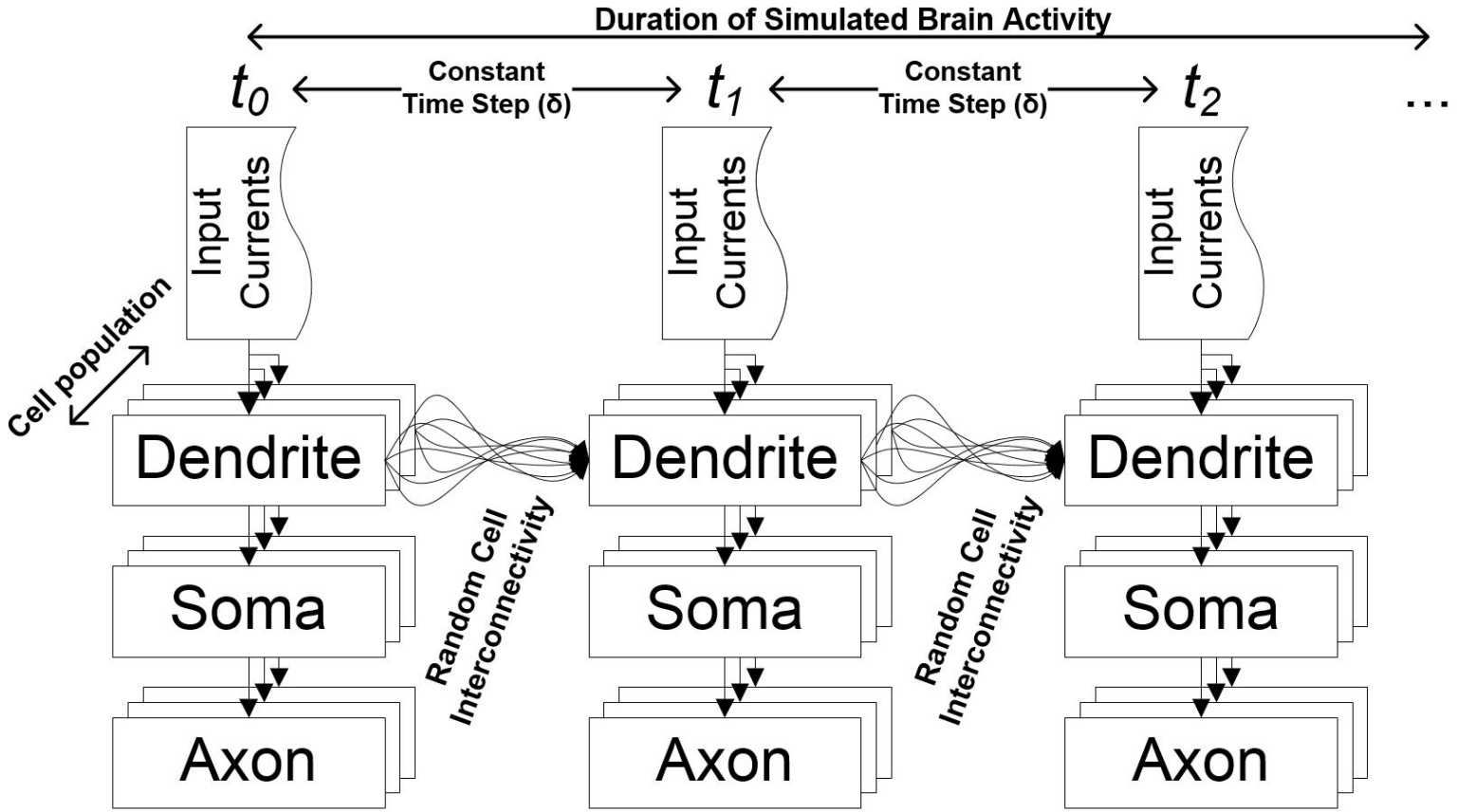
Performance Bottleneck

InfOli Simulator - Description





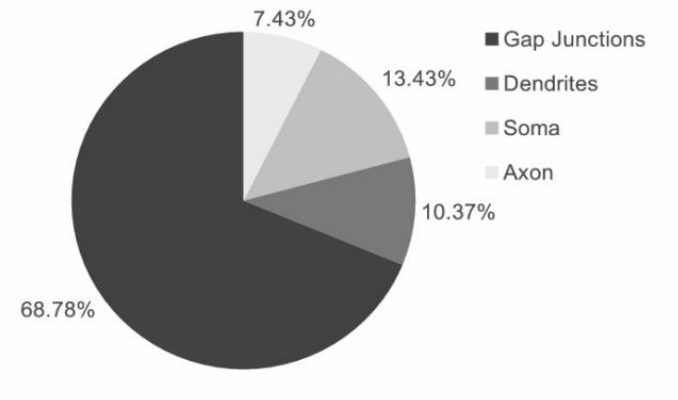
InfOli Simulator - Description



Flowchart of the InfOli Simulator [1]

Infoli Simulator - Serial Code Profiling

- GProf performance analysis tool



Computation time taken by the various compartments in the InfOli model [2]

Computation	FP operations/neuron
Gap junction	12 per connection
Cell compartment	859
I/O and storage	**FP operations/neuron**
Neuron states	19
Evoked input	1
Connectivity vector	1 per connection
Neuron conductances	20
Axon output	1 (Axon voltage)
Neuron computation task	**% of FP ops for 96 cells**
Compartmental computations	43
Gap junctions	57
Computations per step:	$859 * N + 12 * N^2 * C$ C: connectivity density N: network size

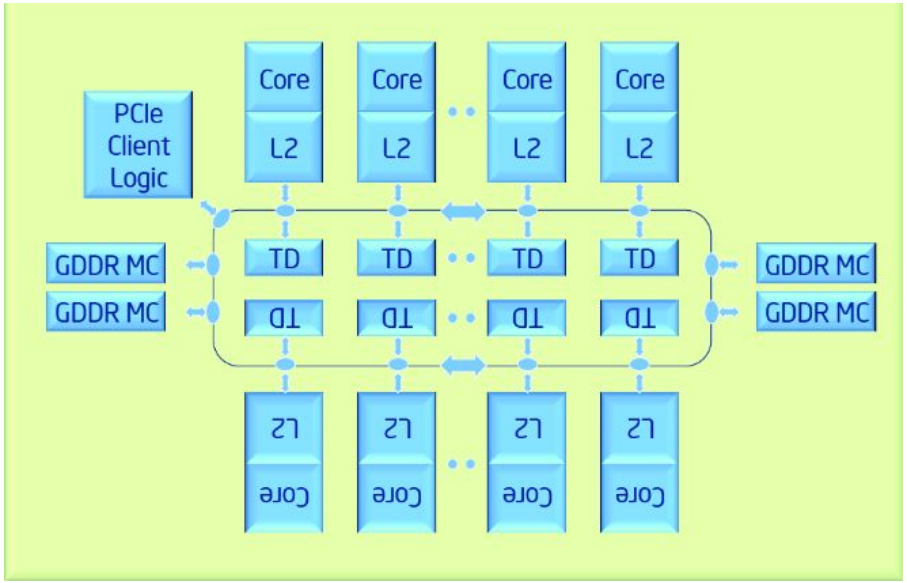
Neuron requirements per simulation step [3]

```
1       /* Main loop: every iteration is a simulation timestep */
2       for(simStep=0;simStep<total_simulation_steps;simStep++) {
                                        .
                                        .
                                        .
3               /*Loop to iterate throught the different neurons and compute every compartment*/
4               for (target_cell=0;target_cell<cellCount;target_cell++) {
5                       /* we simulate a hardcoded input pulse here that differs from step to step */
6                       cellParamsPtr[target_cell].iAppIn = iApp;
7                       cellParamsPtr[target_cell].prevCellState = &cellPtr[simulation_array_ID][target_cell];
8                       cellParamsPtr[target_cell].newCellState = &cellPtr[simulation_array_ID^1][target_cell];
9
10                      CompDend(&cellParamsPtr[target_cell], 0);
11                      CompSoma(&cellParamsPtr[target_cell]);
12                      CompAxon(&cellParamsPtr[target_cell]);
                                        .
                                        .
                                        .
                }

        }
```

```
1   /*Function that models the GJ mechanism. On every timestep, this function has to be called for each neuron*/
2   mod_prec IcNeighbors(mod_prec *neighVdend, mod_prec *neighConductances, mod_prec prevV_dend, int neighbors){

3               int i;
4               mod_prec f, V, Cond, I_c=0;
5               for(i=0;i<neighbors;i++){
6               V = prevV_dend - neighVdend[i];
7               f = 0.8 * exp(-1*pow(V, 2)/100) + 0.2;  /*Exponential functions are costly*/
8               Cond = neighConductances[i];
9               I_c = I_c + (Cond * f * V);
            }
10          return I_c;
    }
```

Porting InfOli to Knights Corner Coprocessor

- Intel Many Integrated Core (MIC) Architecture
  - Highly parallel workloads
  - Built to provide general-purpose programming environment
- KNC:
  - 61 cores (4 threads per core)
  - One VPU per core: 512-bit SIMD instruction set (16 single-precision (SP) or 8 double-precision (DP) operations per cycle)
  - Fully coherent private L2 cache per core
  - Ring interconnect
  - Traditional parallel-programming paradigms



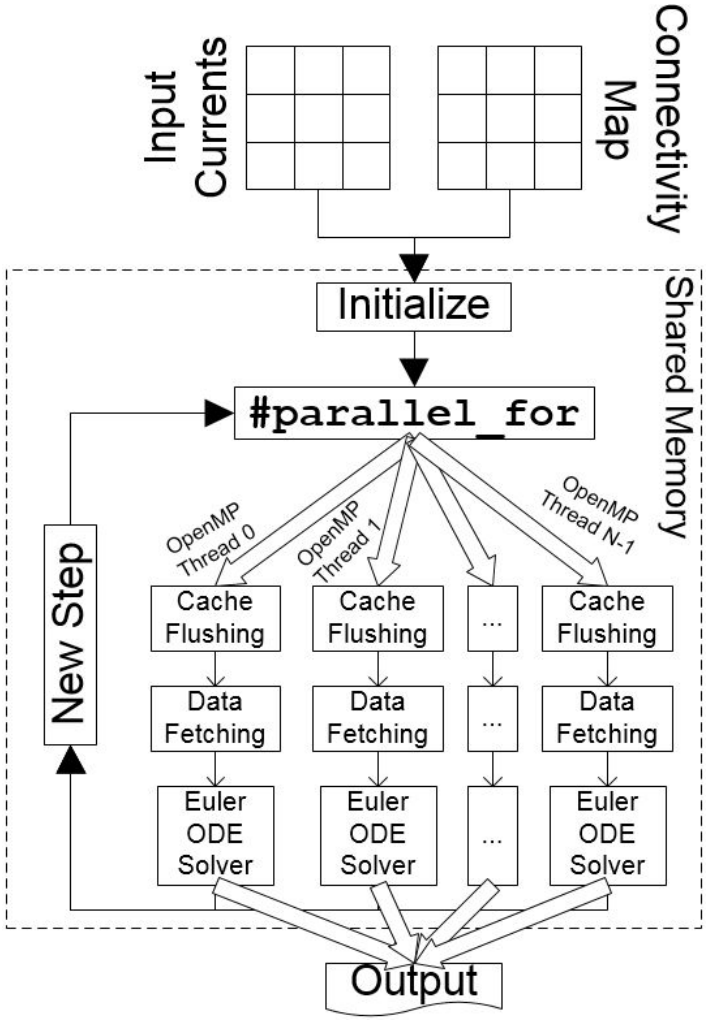
Intel Xeon Phi KNC architecture layout

InfOli Simulator - Parallelization

- Data Partitioning using OpenMP:
  - Each thread can handle a subnetwork
  - Network divided as evenly as possible

- Need for data exchange between threads (on every simulation step)

- Neurons are calculated independently:
  - Threads operate in parallel
  - Each thread vectorizes calculation (“Second level of parallelism”)

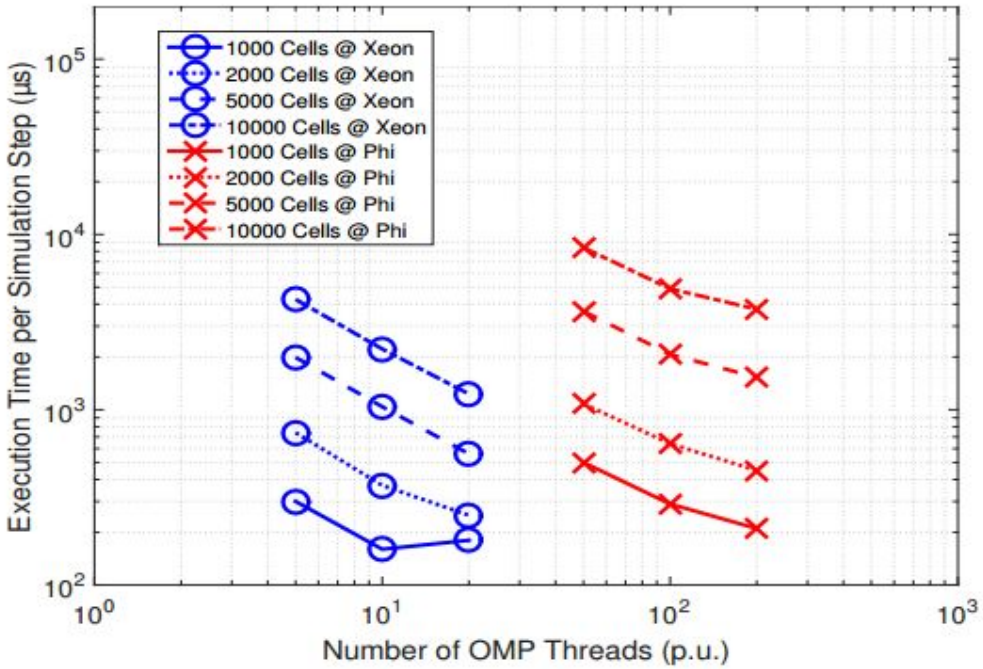


```
1 /* Main loop: every iteration is a simulation timestep */
2 for(simStep=0;simStep<total_simulation_steps;simStep++) {

3 /*Loop to iterate throught the different neurons and compute every compartment*/
4 #pragma omp parallel for simd shared (cellParamsPtr, V_dend, Hcurrent_q, Calcium_r, Potassium_s, I_CaH, Ca2Plus,\
        iApp, iAppIn, I_c, V_soma, g_CaL, Sodium_m, Sodium_h, Calcium_k, Calcium_l, Potassium_n, Potassium_p,
      Potassium_x_s,  V_axon, Sodium_m_a, Sodium_h_a, Potassium_x_a, pOutFile) \
      private(target_cell, tempbuf, q_inf, tau_q, dq_dt, alpha_r, beta_r, r_inf, tau_r, dr_dt, alpha_s, beta_s, s_inf, tau_s,
      ds_dt, dCa_dt, I_sd, I_CaH_temp, I_K_Ca, I_Id, I_h, dVd_dt, k_inf, l_inf, tau_k, tau_l, dk_dt, dl_dt, m_inf, h_inf,
      tau_h, dh_dt, n_inf, p_inf, tau_n, tau_p, dn_dt, dp_dt, alpha_x_s, beta_x_s, x_inf_s, tau_x_s, dx_dt_s, I_ds, I_CaL,
      I_Na_s, I_ls, I_Kdr_s, I_K_s, I_as, dVs_dt, m_inf_a, h_inf_a, tau_h_a, dh_dt_a, alpha_x_a, beta_x_a, x_inf_a,
      tau_x_a, dx_dt_a, I_Na_a, I_la, I_sa, I_K_a, dVa_dt)  firstprivate(simulation_array_ID)
5       for (target_cell=0;target_cell<cellCount;target_cell++) {
6                       /* we simulate a hardcoded input pulse here that differs from step to step */

7                       CompDend(&cellParamsPtr[target_cell], 0);
8                       CompSoma(&cellParamsPtr[target_cell]);
9                       CompAxon(&cellParamsPtr[target_cell]);


                }

}
```

```
1   #pragma omp parallel for shared (cellParamsPtr, iApp, iAppIn, V_dend, I_c, pOutFile) private(target_cell, i, requested_neighbour, f, V, \
    voltage,I_c_storage) firstprivate(simulation_array_ID)
2   for (target_cell=0;target_cell<cellCount;target_cell++) {
                                                                .
                                                                .
                                                                .
     /*  Gathering of the data concerning Vdend of neighours, then computing and storing the incoming Ic from neighbours */
3                       I_c_storage = 0;
4                        __assume_aligned(cellParamsPtr.neighId[target_cell], 64);
5                        __assume_aligned(cellParamsPtr.neighConductances[target_cell], 64);
6                        __assume_aligned(V_dend, 64);
                         #pragma ivdep
7                        for (i=0; i<cellParamsPtr.total_amount_of_neighbours[target_cell]; i++){
8                                requested_neighbour = cellParamsPtr.neighId[target_cell][i];
9                                voltage = V_dend[requested_neighbour];
10                               V = V_dend[target_cell] - voltage;
11                               f = 0.8f * expf(-1*powf(V, 2)/100) + 0.2f;
12                               I_c_storage += cellParamsPtr.neighConductances[target_cell][i] * f * V;
                        }
13                      I_c[target_cell] = I_c_storage;
            }
```

KNC Performance Results

- Intel Xeon E5-2697v2 processor vs. Intel Xeon Phi 5110P accelerator
- 5 s of simulated neural activity (step = 50μs)
- Connectivity created by probability-based generator



OpenMP performance on Xeon and Xeon Phi [1]

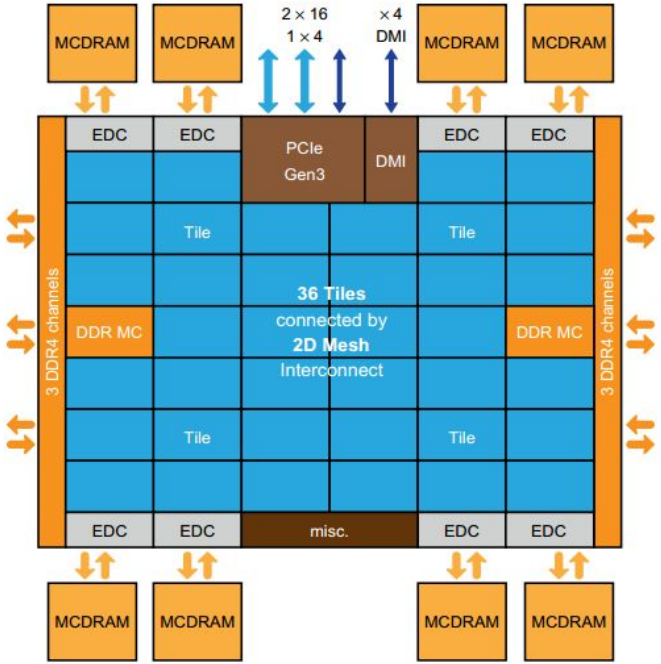
From Knights Corner to Knights Landing

Intel's 1st Generation Xeon Phi: Knights Corner Coprocessor Card

Model: 3120p

Intel's 2nd Generation Xeon Phi: Knights Landing Processor

Model: 7210

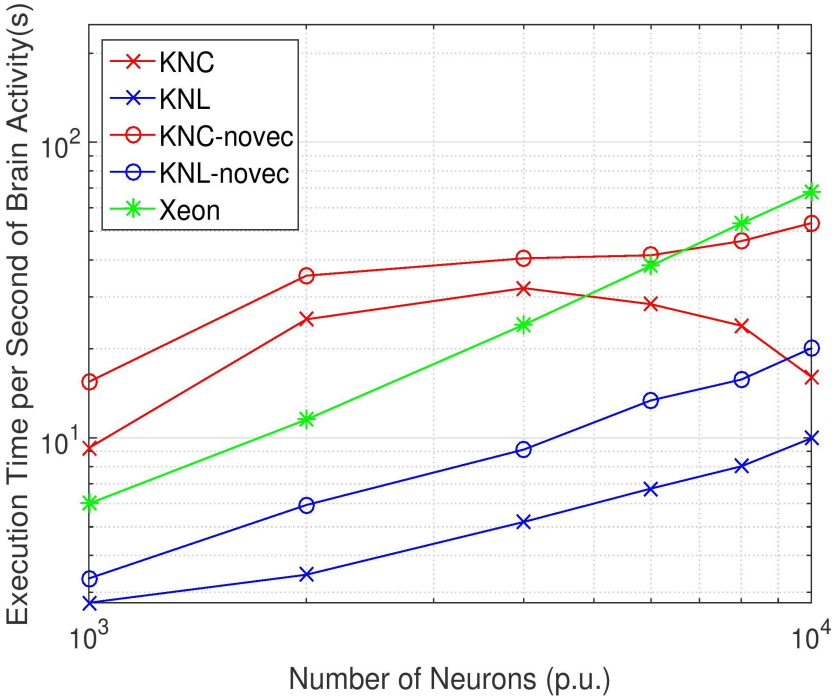


KNL's architecture [4]

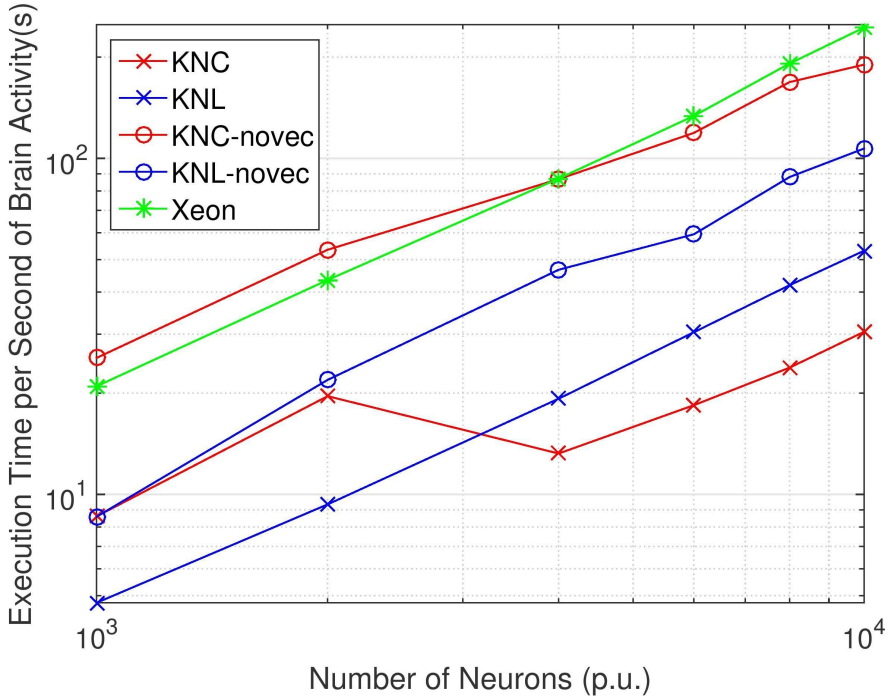
From Knights Corner to Knights Landing: Experimental Evaluation

- *Out-the-box measurements from the KNC on the KNL.*
- Ease of transferring: only recompilation needed
- KNL vs KNC?
  - Better single-threaded performance (3x TFPs)
  - More VPUs, better vectorization support
  - High Bandwidth MCDRAM (set to cache mode)
  - Increased amount of cores, maximum amount of threads
- Experimental evaluation
  - Small (1000) to large (10k) neuron networks
  - Connectivity densities: from 0 up to 1 k GJs per neuron.
  - Exploration of simulation speed, energy used and thread efficiency.

Results - Execution time

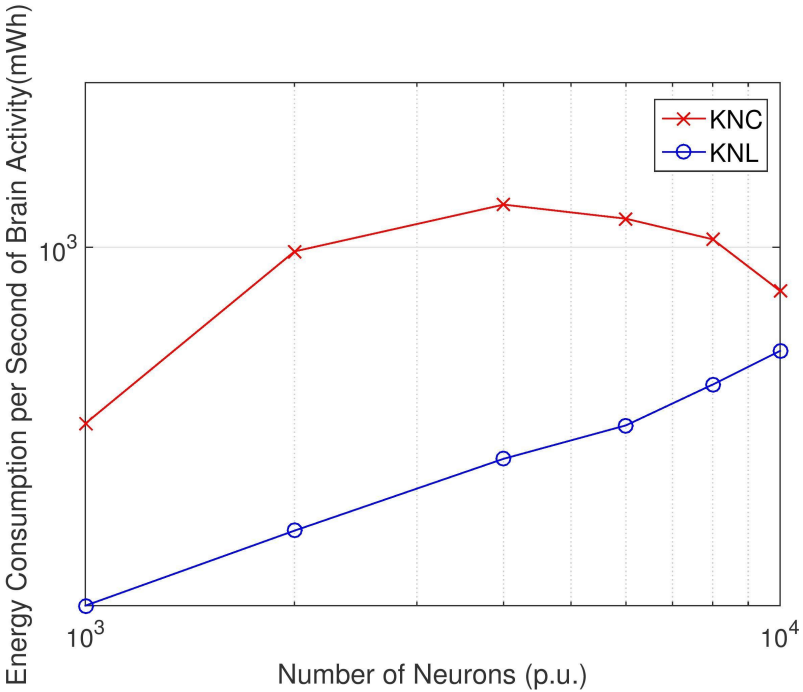


Low-density networks

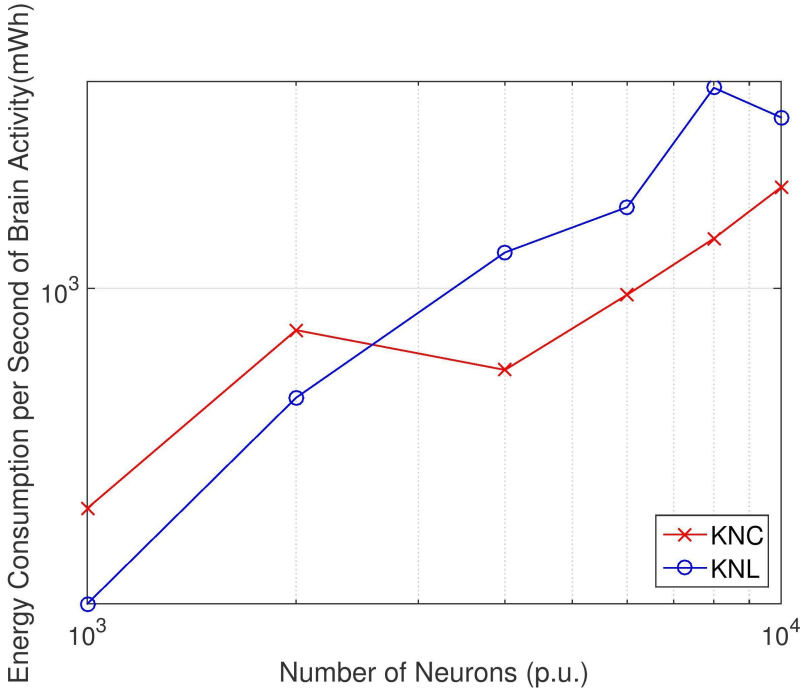


High-density networks

Results - Energy Consumption

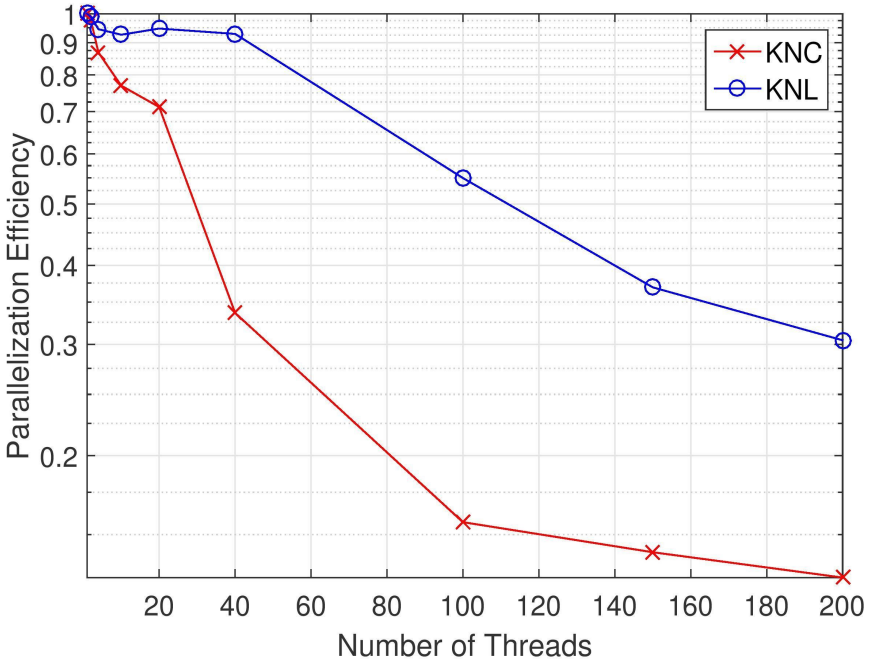


Low-density networks

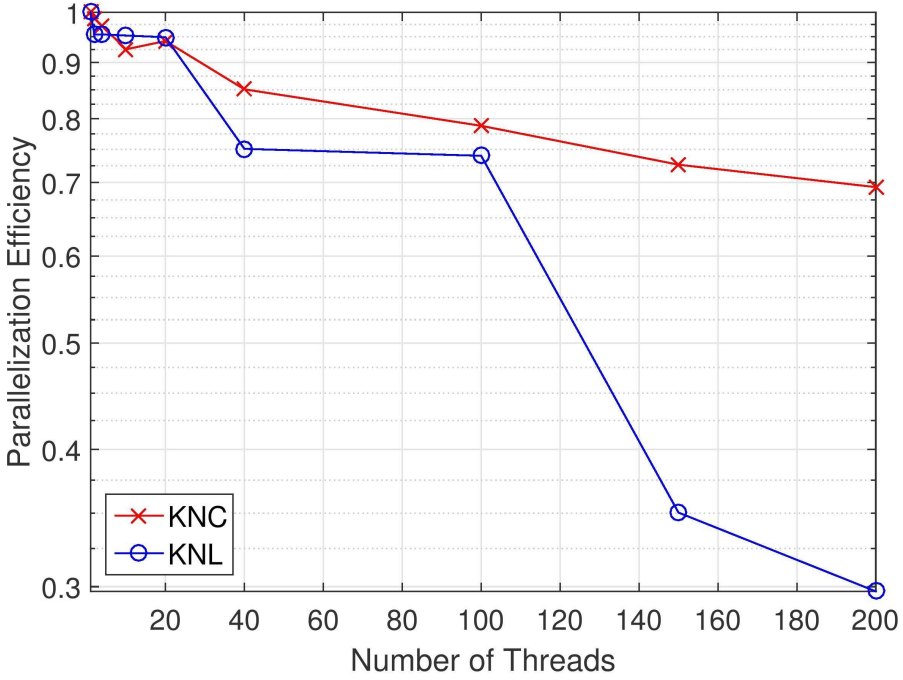


High-density networks

Results - Efficiency



High-density network of 1k neurons



High-density network of 10k neurons

Results - Analysis

- Sparse networks are more serial in nature, so they operate well on KNL (superior single-threaded performance).
- Denser networks heavily favor vectorization-enabled implementations:
  - Vectorization on the KNC is significantly better after a certain point.
  - KNL performance is worse for some of the heaviest workloads.
- KNL´s lower TDP leads to significant energy gains.
  - Gap lessens with higher workload.
  - On heavier workloads, KNL´s lower TDP offset by increased simulation times.
- KNL very efficient for 1 thread per core, however efficiency takes a significant hit past 100 threads.
- KNC retains acceptable efficiency for 200 threads.

Conclusions and Insights

- On average, 2.4x speedup, comparable to expected single thread performance upgrade of KNL over KNC (3x).
- Lower TDP leads to overall energy savings (~50%) on KNL. Up to 75% saving on low density networks!
- Thread efficiency suffers on the KNL possibly because of lack of fine-tuning of the application to the architectural details of the platform.
- Best practice suggests ~2 threads per KNL core.
- KNL displays greater predictability in performance.

References

[1] Chatzikonstantis, G., Rodopoulos, D., Nomikou, S., Strydis, C., De Zeeuw, C. I., & Soudris, D. (2016, May). First impressions from detailed brain model simulations on a Xeon/Xeon-Phi node. In *Proceedings of the ACM International Conference on Computing Frontiers* (pp. 361-364). ACM.

[2] Smaragdos, G., Isaza, S., van Eijk, M. F., Sourdis, I., & Strydis, C. (2014, February). FPGA-based biophysically-meaningful modeling of olivocerebellar neurons. In *Proceedings of the 2014 ACM/SIGDA international symposium on Field-programmable gate arrays* (pp. 89-98). ACM.

[3] Smaragdos, G., Chatzikonstantis, G., Nomikou, S., Rodopoulos, D., Sourdis, I., Soudris, D., ... & Strydis, C. (2016, April). Performance analysis of accelerated biophysically-meaningful neuron simulations. In *Performance Analysis of Systems and Software (ISPASS), 2016 IEEE International Symposium on* (pp. 1-11). IEEE.

[4] Jeffers, J., Reinders, J., & Sodani, A. (2016). *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.

[5] Chatzikonstantis, G., Jiménez, D., Meneses, E., Strydis, C., Sidiropoulos, H., & Soudris, D. (2017, June). From Knights Corner to Landing: A Case Study Based on a Hodgkin-Huxley Neuron Simulator. In *International Conference on High Performance Computing* (pp. 363-375). Springer, Cham.