

A High-Productivity Language for Computational Science

Josh Milthorpe

Productivity

Challenges for high-performance computing

The standard approach in high-performance computing is currently to use C++ or FORTRAN for standard sequential programming, combined with MPI for parallelism.

Problems with the standard approach:

- readability
- correctness
- fragmented view of computation
- lower productivity

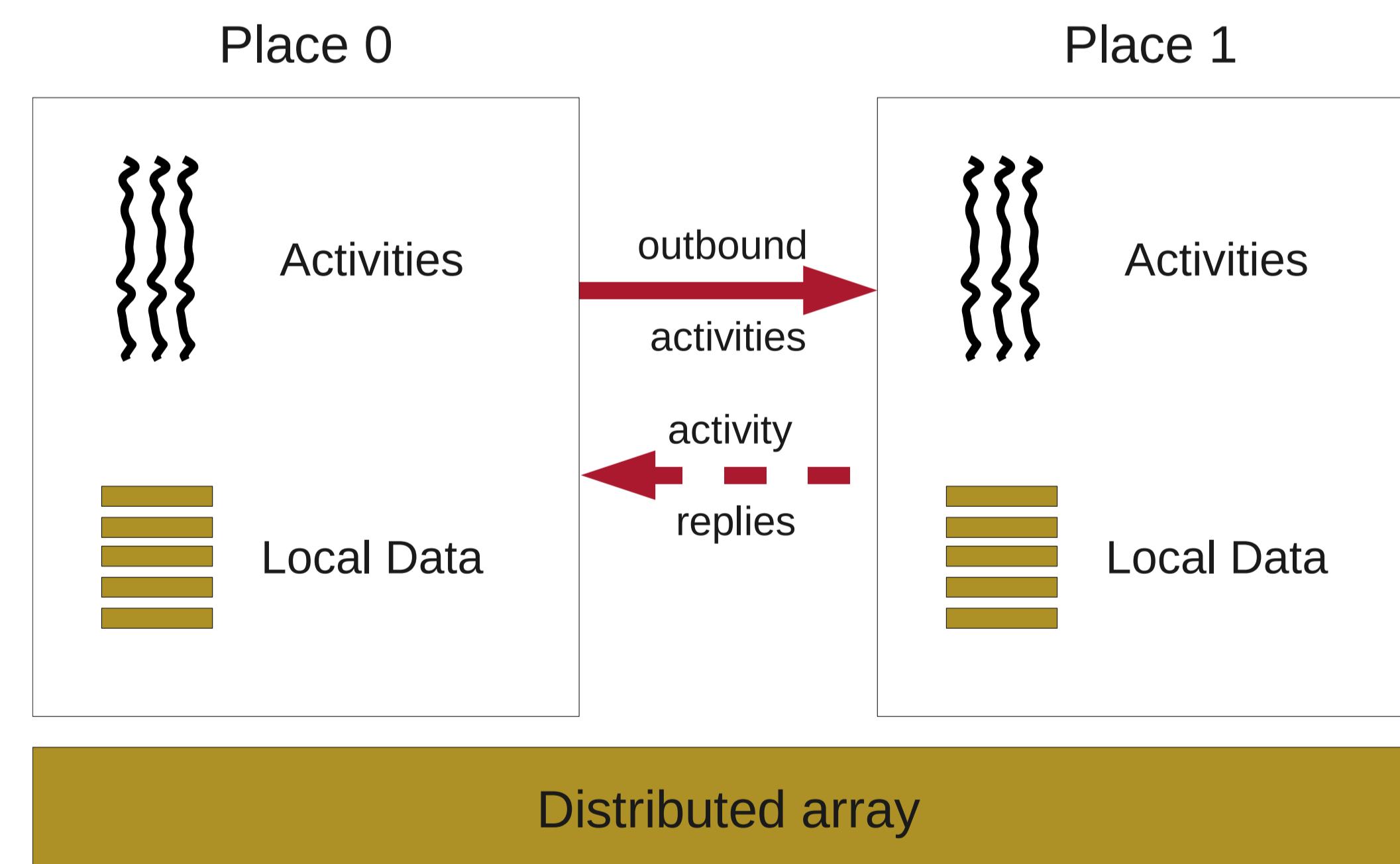
The DARPA High Productivity Computing Systems Program^[1] aims to increase the value and reduce the cost (in terms of hardware, software development, maintenance or compute time) of high-performance computing. One theme of the program is the creation of **new programming languages** for emerging **parallel architectures**. Three languages have been created: Fortress, Chapel, and X10.

X10

A new language for high-productivity computing

X10^[2] is a managed object-oriented language, based on a serial subset of Java with the addition of explicit localisation through **places** and dynamic, asynchronous **activities** for concurrency. It supports a variety of common parallel programming idioms.

X10 provides a rich **array sublanguage**. An array has a *region* (the set of index points) and a *distribution* specifying the location (in memory) of each point in the region. Regions and distributions provide the foundation for parallel constructs such as *ateach* which executes a statement over all the points in an array distribution.



X10 places are collections of activities and the data upon which they operate.

[1] <http://highproductivity.org/>

[2] Saraswat, V., and Nystrom, N. Report on the experimental language X10 version 1.7. Tech. rep., IBM, May 2009. <http://dist.codehaus.org/x10/documentation/languagespec/x10-174.pdf>

General matrix multiplication: Standard approach vs. X10

C++ / MPI

```
matmul() {
  MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
  MPI_Comm_size(MPI_COMM_WORLD, &numtasks);

  if (taskid == 0) {
    for (i = 1; i < numtasks; i++) {
      MPI_Send(&a[i*N/numtasks][0], N*N/numtasks,
              MPI_INT, i, 0, MPI_COMM_WORLD);
      MPI_Send(b, N*N, MPI_INT,
              i, 0, MPI_COMM_WORLD);
    }
  } else {
    MPI_Recv(a, N*N/numtasks, MPI_INT,
            0, 0, MPI_COMM_WORLD, 0);
    MPI_Recv(b, N*N, MPI_INT,
            0, 0, MPI_COMM_WORLD, 0);
  }

  for (i = 0; i < N/numtasks; i++) {
    for (j = 0; j < N; j++) {
      c[i][j] = 0;
      for (k = 0; k < N; k++) {
        c[i][j] += a[i][k] * b[k][j];
      }
    }
  }

  if (taskid != 0) {
    MPI_Send(c, N*N/numtasks, MPI_INT, 0, 0,
            MPI_COMM_WORLD);
  } else {
    for (i = 1; i < numtasks; i++) {
      MPI_Recv(&c[i*N/numtasks][0], N*N/numtasks,
              MPI_INT, i, 0, MPI_COMM_WORLD, 0);
    }
  }
}
```

X10

```
def matmul() {
  val r = [0..N-1,0..N-1] as Region;

  val a = Array.make[double](
    Dist.makeBlock(r, 0));
  val b = Array.make[double](r);
  val c = Array.make[double](
    Dist.makeBlock(r, 0));

  finish ateach (val (i): Point in a) {
    for (var j:int=0; j<N; j++) {
      c(i,j) = 0.0;
      for (var k:int=0; k<N; k++) {
        c(i,j) += a(i,k)*b(k,j);
      }
    }
  }
}
```

Research questions

- What aspects of the X10 language specification, compilers and runtimes are critical for performance?
- How does the performance of X10 compare with the standard approach (FORTRAN / C++ and MPI) for biochemical applications?
- How can runtime optimization be used to enhance performance?
- How can language features of X10 support good performance across different architectures e.g. CPU vs. GPU?
- How do Particle Mesh Ewald and Fast Multipole methods compare for typical biochemical problems?

Supervised by Alistair Rendell

in collaboration with researchers at IBM T.J. Watson Research Center

Computational Biochemistry

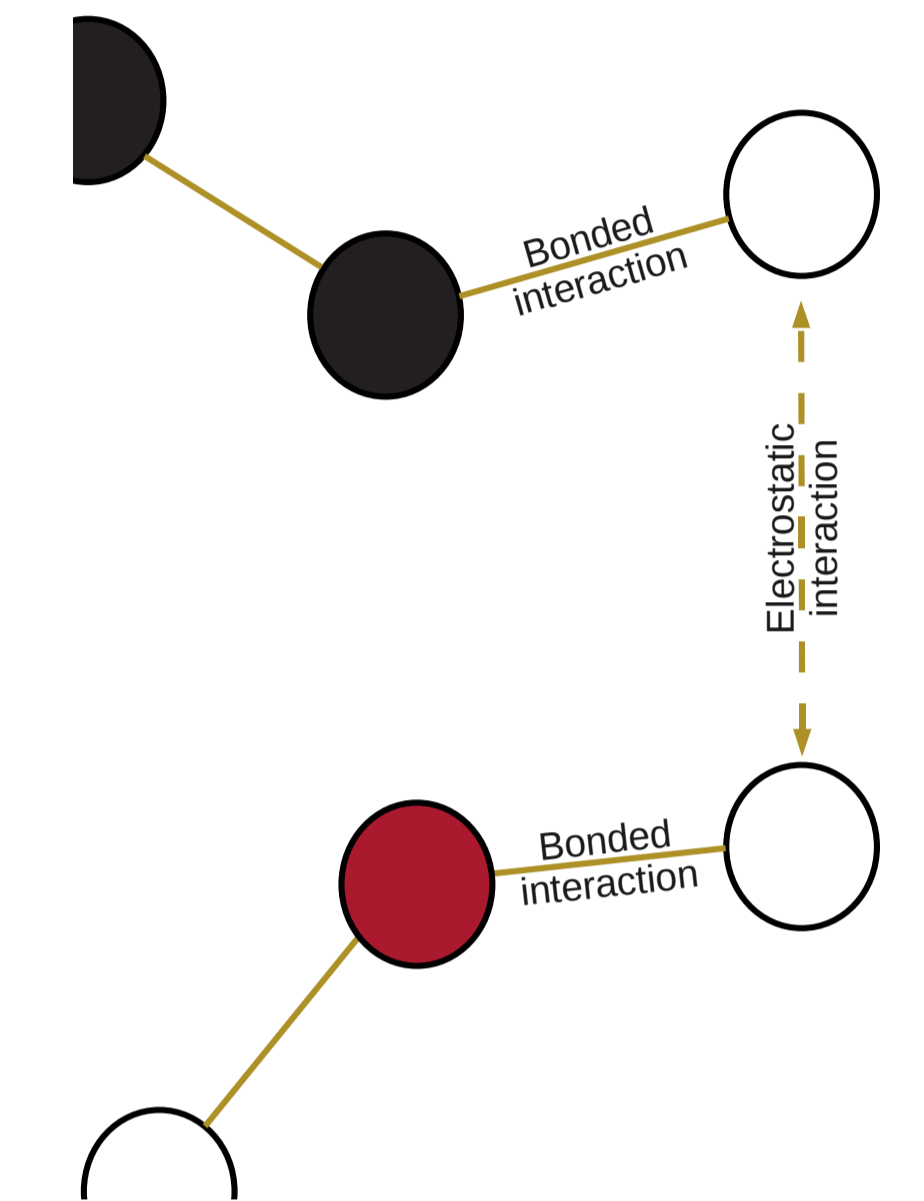
Modeling the building blocks of life

Aim: to develop molecular dynamics codes in X10 using Particle-Mesh Ewald and Fast Multipole methods, and compare performance for typical problems; for example, simulation of the dynamic behaviour of ion channels.

Molecular dynamics is the simulation of dynamical systems of molecules and ions in terms of forces between their constituent atoms. Fundamental to this method is the **force field**, which defines simplified interactions between pairs of atoms in a fixed bonding structure. The force field is used to simulate dynamic behaviour by integrating the forces over time. This approach is used in computational biochemistry to explore processes like protein folding and docking.

Key factors in molecular dynamics:

- Force field representation
- Time integration
- Evaluation of long-range interactions



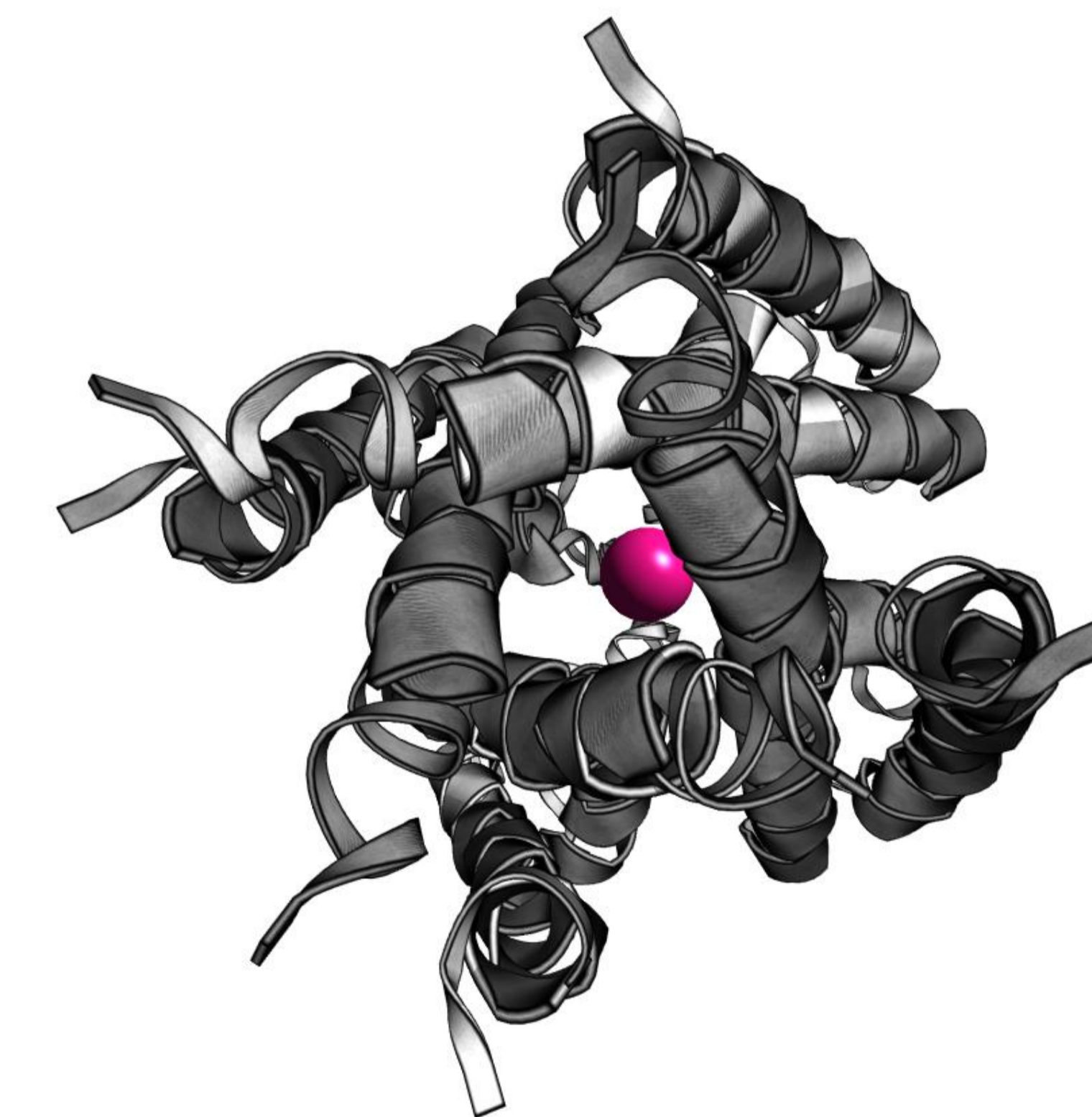
The time-critical part of a molecular dynamics simulation is the calculation of long-range electrostatic interactions.

Electrostatic interactions

The time required to exactly calculate all interactions is proportional to the square of the number of particles. However approximate methods exist with better scaling properties.

Particle Mesh Ewald methods approximate long range potentials over a mesh of grid points in the simulation box. A 3D Fast Fourier Transform is used to solve Poisson's equation for the potential at each grid point. Particle potentials are interpolated between grid points.

Fast Multipole methods use multipole approximations for interactions between widely separated groups of particles. Although in theory these methods scale better than PME, practical implementations have not yet been competitive for biochemical problems.



Ion channels are present in the membranes of all cells and support nerve impulses and other rapid biological processes. This system – the potassium ion channel – contains over 2800 atoms.