Parallel IO on the BlueGene/P

Robert Miller

Gene Network Sciences, Inc.

May 8, 2008

Collaborators: Jeff Fox, Fernando Siso, GNS
Greg Buzzard, Purdue University

Special thanks to: Vitali Morozov, ANL

Problem Description

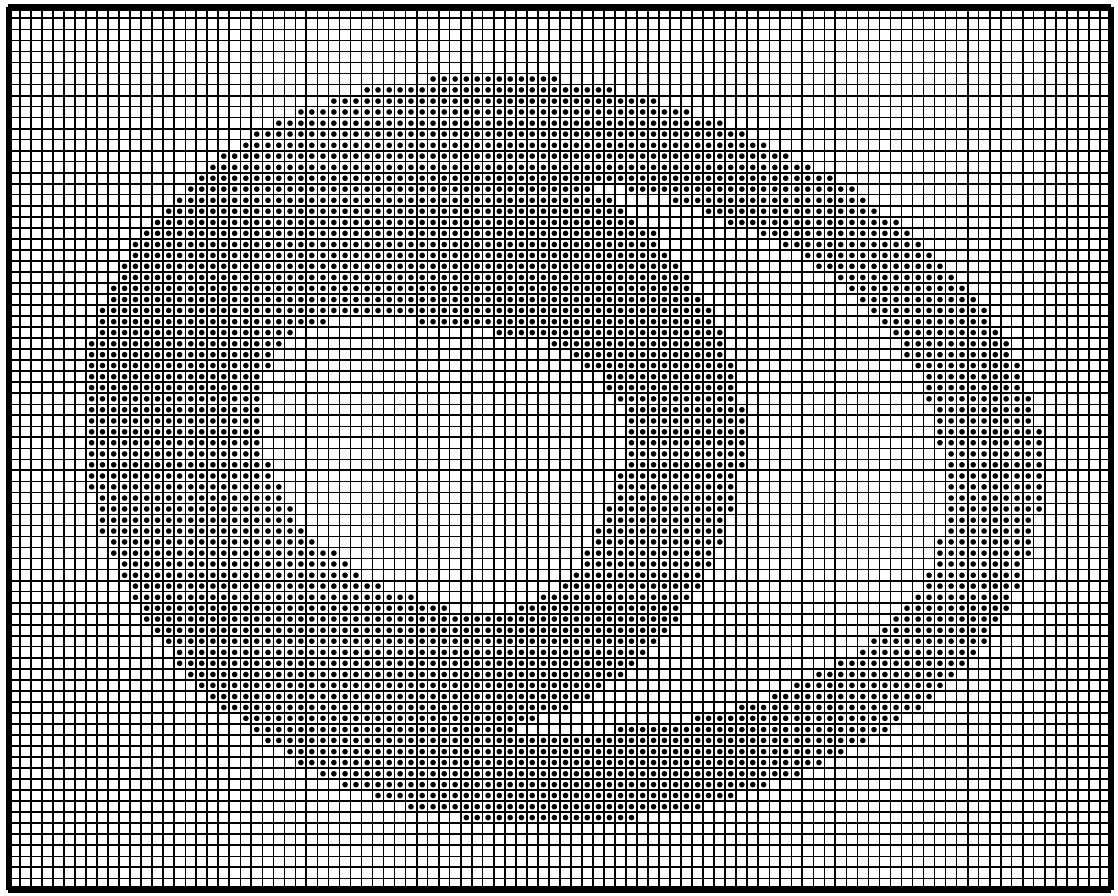
We are modelling the propagation of electrical waves in cardiac tissue.

$$\frac{\partial V}{\partial t} = \nabla \cdot D \nabla V - I_{\text{ion}}$$

$$\vec{n} \cdot D \nabla V = 0 \quad \text{on } \partial R$$

where V is the membrane potential, D is a spatially-varying diffusion tensor (determined from fiber orientation), I_{ion} is the contribution from an ionic model of a heart cell and R is the region occupied by the heart tissue.

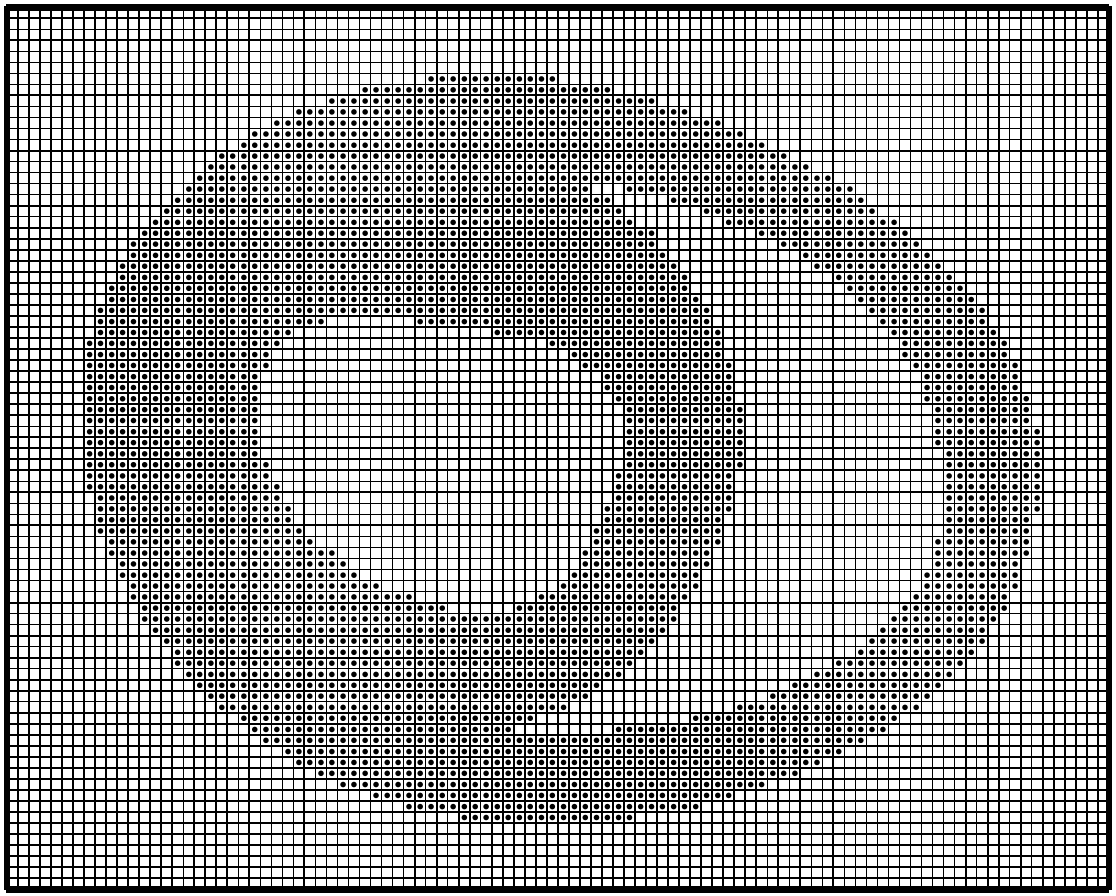
Finite-Difference Formulation



100×80 grid with 3,133 active nodes. CVM has 13 state variables $\Longrightarrow$ $40,729$ unknowns.

Mapping to processors

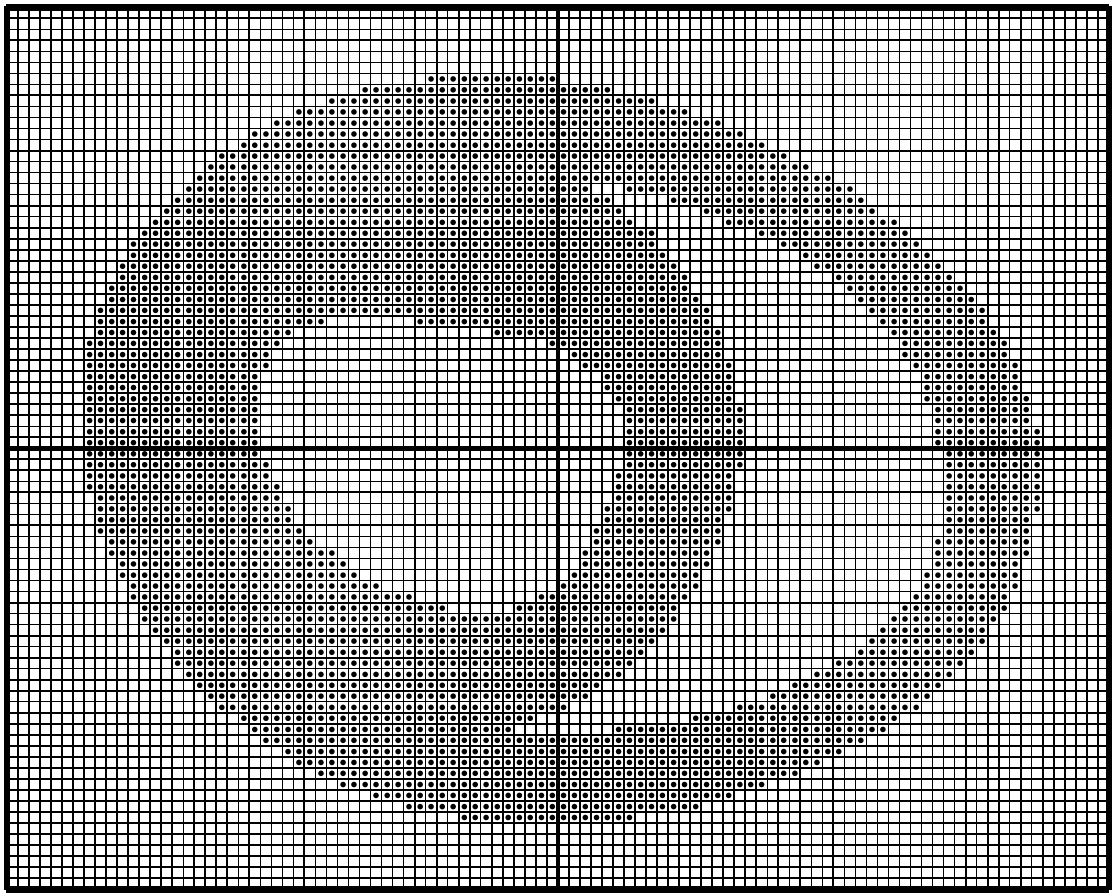
Assume 2×2 torus $\Longrightarrow$ $3133/4 \approx 783$ active nodes/processor.



Mapping to processors

Assume 2×2 torus $\Longrightarrow$ $3133/4 \approx 783$ active nodes/processor.

939



686

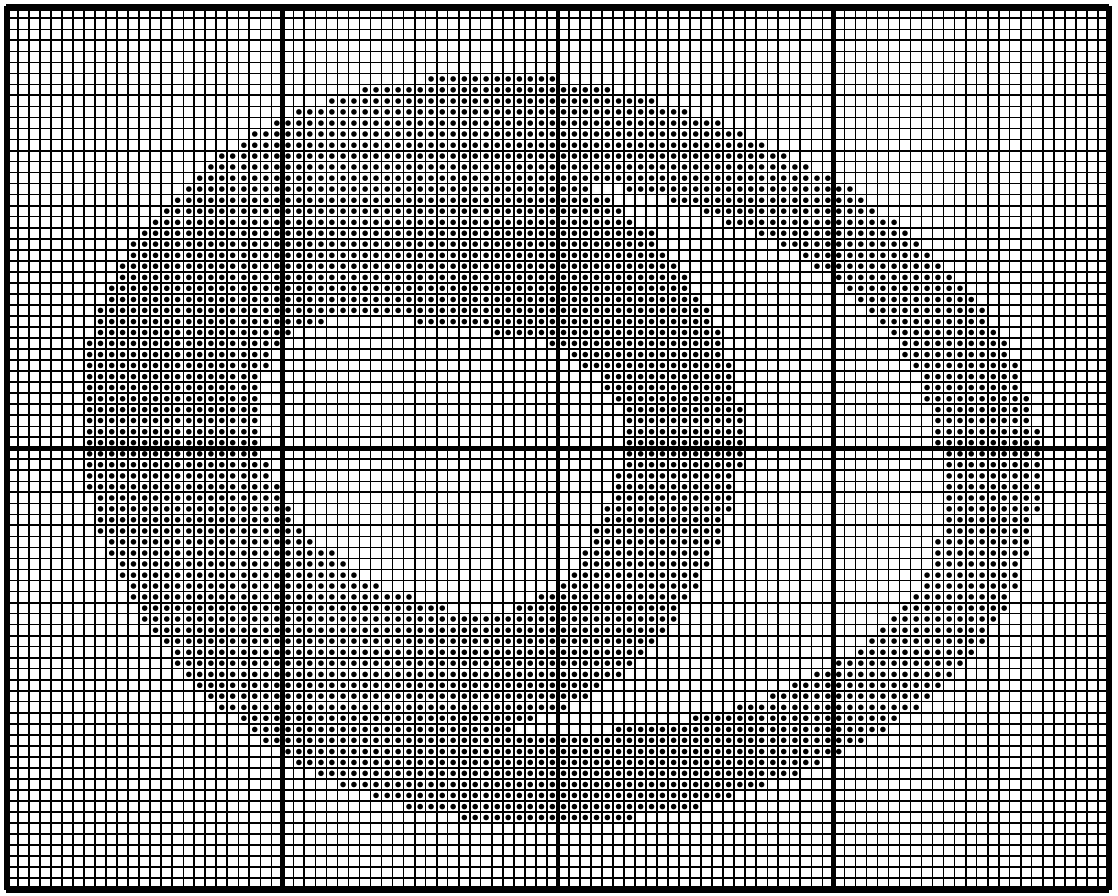
826

682

Mapping to processors

Assume 2×2 torus $\Longrightarrow$ $3133/4 \approx 783$ active nodes/processor.

939



686

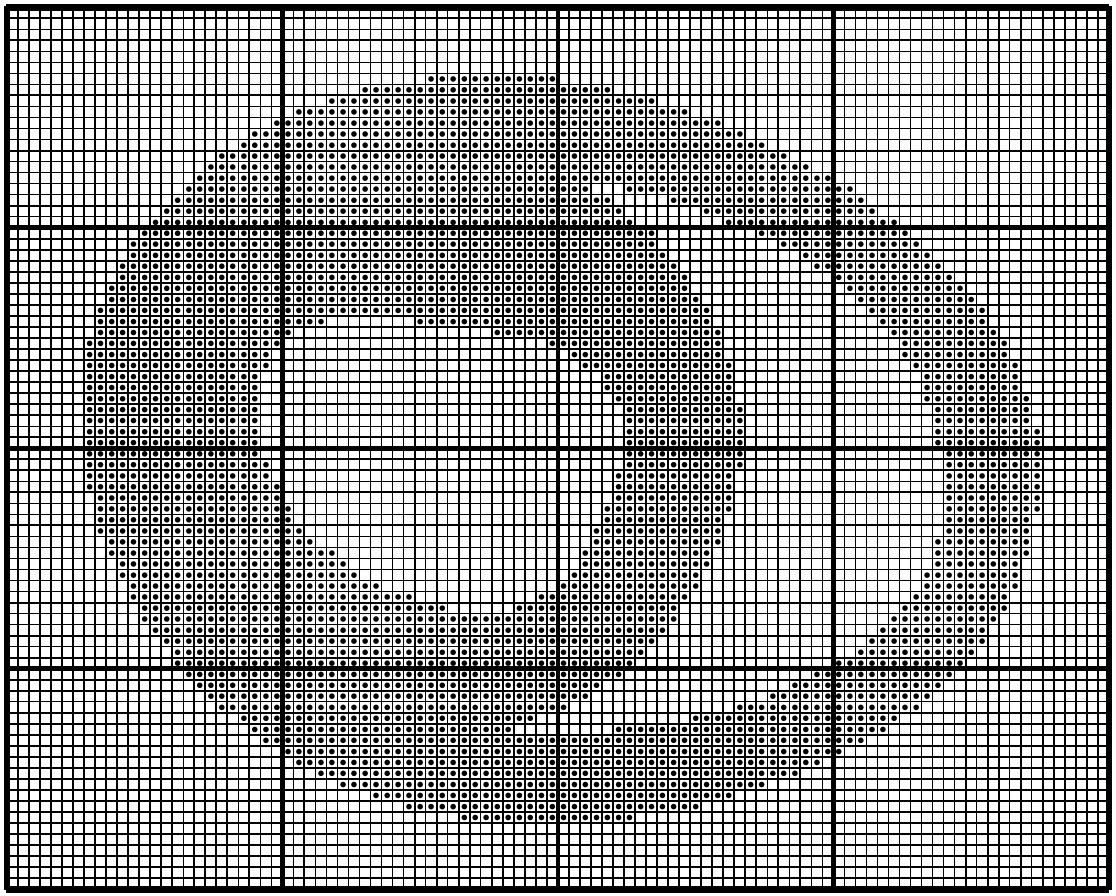
826

682

Mapping to processors

Assume 2×2 torus $\Longrightarrow$ $3133/4 \approx 783$ active nodes/processor.

939



686

826

682

Canine Example

- Grid is $363 \times 279 \times 257$; 26,028,189 total nodes
- 8,623,314 active nodes; 112,103,082 unknowns

Perform 400ms simulation saving two variables (V and diffusion current) at 1ms intervals.

For 2 racks in virtual node mode, target is $8,623,314/8,192 \approx 1053$ active nodes/processor.

Using box size of $5 \times 2 \times 2$, maximum number of active nodes is 1228.

Original Output Scheme

1. Map solution into interval $[-100, 100]$

```
char* acData
for (i=0; i<iLocalMaxNodeIndex; ++i) {
  idxCellModel = Nodes[i]->GetCellModel()->GetIndex();
  rValue = (Nodes[i]->GetCurConds())[viMap[idxCellModel]];
  iOutput = (int) floor(200*(rValue - rLowerCutoff) /
            (rUpperCutoff - rLowerCutoff) - 100);
  iOutput = std::max(iOutput, -100);
  iOutput = std::min(iOutput, 100);
  acData[i] = (char) iOutput;
}
```

2. Transfer data to master

```
MPI_Gatherv(acData, iLocalMaxNodeIndex, MPI_CHAR,
            &(vcDataBuf[0]), &(viDataCount[0]),
            &(viDataDispl[0]), MPI_CHAR, 0, MPI_COMM_WORLD);
```

3. Master puts data in order, compresses array and writes to file system.

Original Output Scheme (cont.)

```
for (int iProc=0; iProc<NumProc; ++iProc) {
  for (int iLocal=0; iLocal<viDataCount[iProc]; ++iLocal) {
    iGlobal = vviProcLocalToGlobalIndex[iProc][iLocal];
    iBuffer = viDataDispl[iProc] + iLocal;
    vcDataOut[iGlobal] = vcDataBuf[iBuffer];
  }
}
```

Timings on 1 rack

Simulation took 992.5 seconds, saving output took 1035.9 seconds. Save called 802 times $\Longrightarrow$ 1.29 seconds/call; compression routine took 0.376 seconds/call

Timings on 2 racks

Simulation took 537.5 seconds (1.34 seconds/1ms simulation)

Save took 2.05 seconds/call (compression was 0.383 seconds/call).

Parallel Output Scheme

1. Open parallel file (all processes)

```
MPI_Info info;
int mode = MPI_MODE_WRONLY | MPI_MODE_CREATE;
MPI_Info_create(&info);
char* filename = "...";
MPI_File fh;
MPI_File_open(MPI_COMM_WORLD, filename, mode, info, &fh);
```

2. Master writes header data (grid size, output times, etc.)

```
MPI_Offset offset = 0;
void* data;
int count;
MPI_Datatype type;
MPI_Status status;
// several calls like this:
MPI_File_write_at(fh, offset, data, count, type, &status);
offset += ...;
```

Parallel Output Scheme (cont.)

3. Master computes buffer size, number of times to call Save between writes and broadcasts to workers.

```
int maxActiveNodes = *std::max_element(viDataCount.begin(),
                                       viDataCount.end());
nSavesPerWrite = maxBuffSize /
                 (maxActiveNodes * numVar * data_size);
MPI_File_write_at(fh,offset,&nSavesPerWrite,1,MPI_INT,&status);
offset += sizeof(int);
buff_size = data_size*numVar*maxActiveNodes*nSavesPerWrite;
int buff[3] = { offset, nSavesPerWrite, buff_size };
MPI_Bcast(buff, 3, MPI_INT, 0, MPI_COMM_WORLD);
```

4. All processes save mapping data in header

```
MPI_Offset my_offset = offset+viDataDispl[myRank]*sizeof(int);
MPI_File_write_at_all(fh, my_offset, &viLocalToGlobalIndex[0],
                      viDataCount[myRank],MPI_INT,&status);
offset += viDataDispl.back() * sizeof(int);
```

Parallel Output Scheme (cont.)

5. Each process allocates buffer using `buff_size`
6. Each time SaveOutputData is called, each process copies the current solution into its output buffer

```
int bytesPerVariablePerSave = data_size * iLocalMaxNodeIndex;
char* pBuff = (char*)output_buffer +
                bytesPerVariablePerSave*numOutputVar*numSaves;
for (int ivar=0; ivar<numOutputVar; ++ivar) {
  vint viMap = vpOutputVariable[ivar]->GetCellModelToIndexMap();
  for (int i=0; i<iLocalMaxNodeIndex; ++i) {
    int idx = Nodes[i]->GetCellModel()->GetIndex();
    fData[i] = (float)(Nodes[i]->GetCurConds())[viMap[idx]];
  }
  memcpy(pBuff, (char*)fData, bytesPerVariablePerSave);
  pBuff += bytesPerVariablePerSave;
}
++numSaves;
```

Parallel Output Scheme (cont.)

7. All processes write data to file

```
if (numSaves == numSavesPerWrite || endOfRun) {
  int count = numSaves * numOutputVar * viDataCount[myRank];
  MPI_Offset my_offset = offset +
    (MPI_Offset)(data_size*numSaves*numOutputVar
                 *viDataDispl[myRank]);
  MPI_Status status;
  MPI_File_write_at_all(fh, my_offset, output_buffer,
                        count, MPI_FLOAT, &status);
  offset += (MPI_Offset)(viDataDispl[numProc]*data_size
                         *numSaves*numOutputVar);
  numSaves = 0;
}
```

Timing for 2 rack run

Saved 26GB in 11.26 sec.