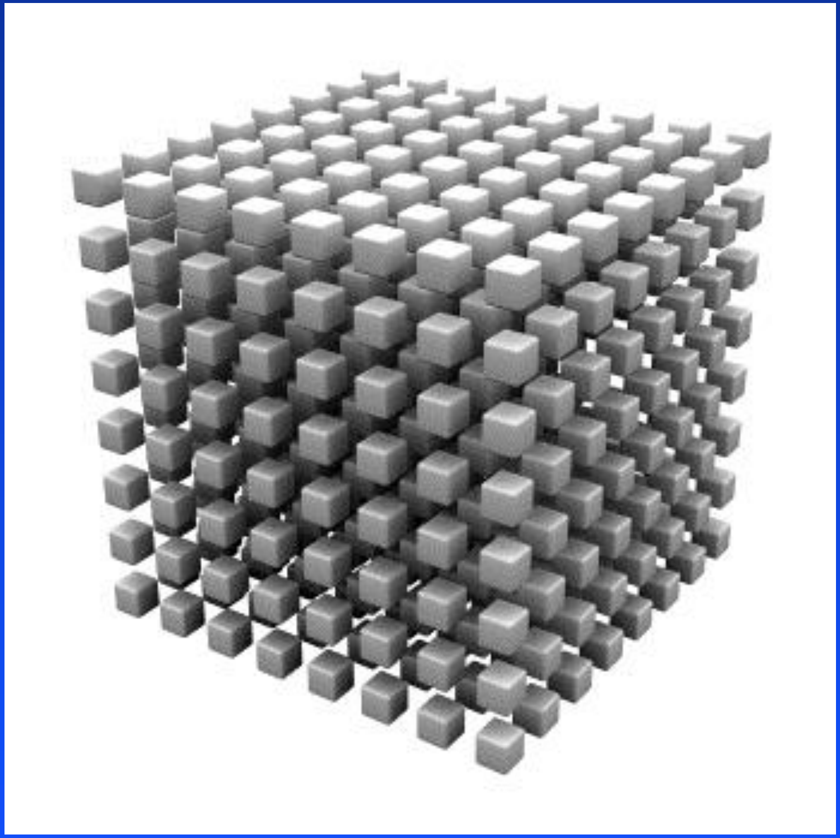
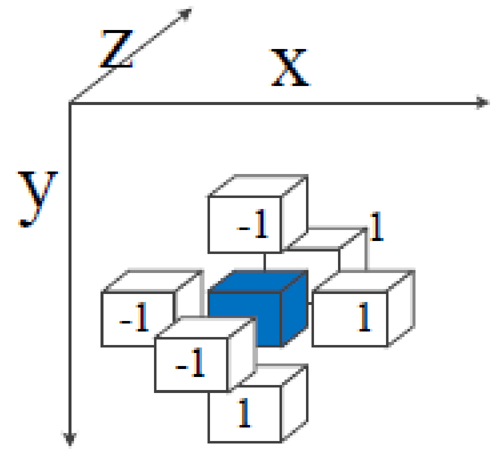
Jacobi in node





(a) 7-point Jacobi

Sequential Code

```
for( i = 1; i < X+1; i ++ ) {
for( j = 1; j < Y+1; j ++ ) {
for( k = 1; k < Z+1; k ++){

    matrixB[k+j*rZ+i*rY*rZ] =
            c1*matrixA[k+j*rZ+i*rY*rZ] +
            c2*(matrixA[k+j*rZ+(i-1)*rY*rZ] +
                matrixA[k+j*rZ+(i+1)*rY*rZ] +
                matrixA[k+(j-1)*rZ+i*rY*rZ] +
                matrixA[k+(j+1)*rZ+i*rY*rZ] +
                matrixA[(k-1)+j*rZ+i*rY*rZ] +
                matrixA[(k+1)+j*rZ+i*rY*rZ]);
}
}
}
```

How to Tune the performance

- OpemMP
- OpenMP + Vectorization
- OpenMP + Vec + Blocking

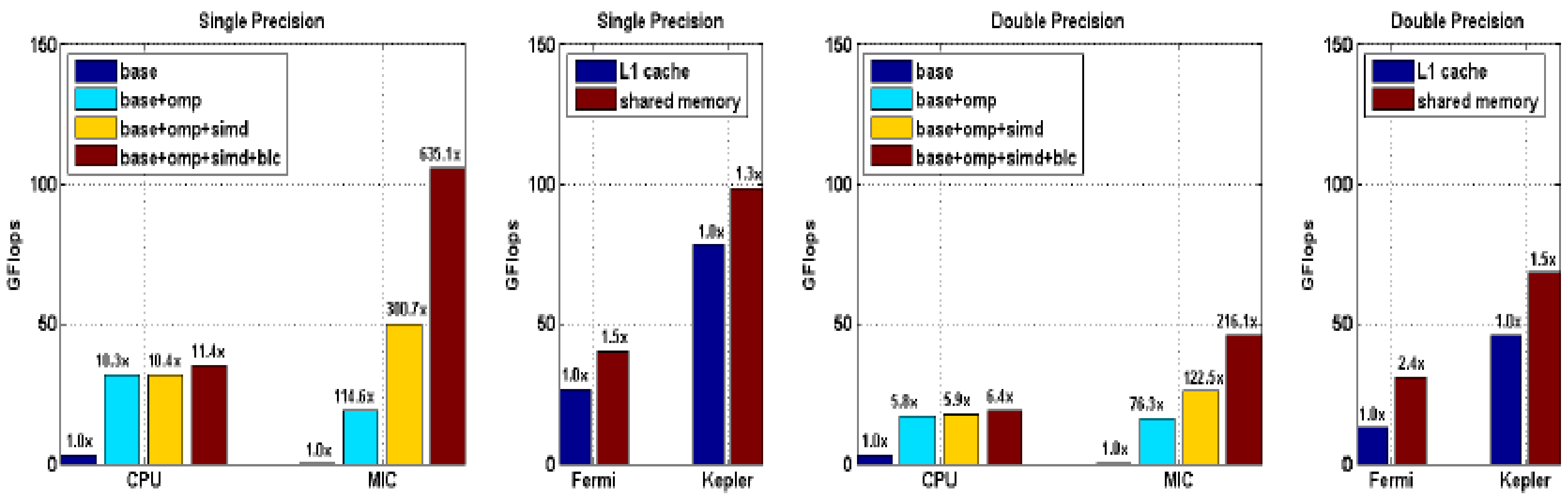
Do you remember?
Using a Simple Model of Memory to Optimize

- Assume just 2 levels in the hierarchy, fast and slow
- All data initially in slow memory
  - m = number of memory elements (words) moved between fast and slow memory
  - t_m = time per slow memory operation
  - f = number of arithmetic operations
  - t_f = time per arithmetic operation $<< t_m$
  - q = f / m average number of flops per slow memory access
- Minimum possible time = $f * t_f$ when all data in fast memory
- Actual time
  - $f * t_f + m * t_m = f * t_f * (1 + \boxed{t_m/t_f} * 1/q)$
- Larger q means time closer to minimum $f * t_f$
  - $q \geq t_m/t_f$ needed to get at least half of peak speed

Computational Intensity: Key to algorithm efficiency

Machine Balance: Key to machine efficiency

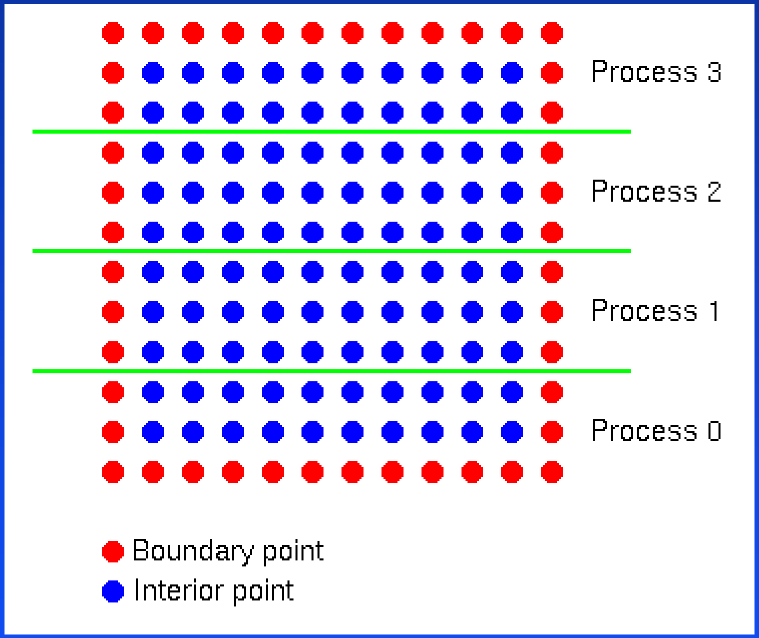
Our Results on SandyBridge, MIC and Kepler



(a) Jacobi

Jacobi Iteration for MPI

- Simple parallel data structure



- Processes exchange rows with neighbors

Background to Tests

- Goals
  - » Identify better performing idioms for the same communication operation
  - » Understand these by understanding the underlying MPI process
  - » Provide a starting point for evaluating additional options (there are many ways to write even simple codes)

Send and Recv

- Simplest use of send and recv

```
{
   MPI_Status status;
   MPI_Comm ring_comm = mesh->ring_comm;

   /* Send up, then receive from below */
   MPI_Send( xlocal + maxm * lrow, maxm, MPI_DOUBLE, up_nbr, 0,
             ring_comm );
   MPI_Recv( xlocal, maxm, MPI_DOUBLE, down_nbr, 0, ring_comm, &status )
   /* Send down, then receive from above */
   MPI_Send( xlocal + maxm, maxm, MPI_DOUBLE, down_nbr, 1, ring_comm );
   MPI_Recv( xlocal + maxm * (lrow + 1), maxm, MPI_DOUBLE, up_nbr, 1,
             ring_comm, &status );
}
```

Better to start receives first

- Irecv, Isend, Waitall

```
MPI_Status statuses[4];
MPI_Comm ring_comm;
MPI_Request r[4];

   /* Send up, then receive from below */
   MPI_Irecv( xlocal, maxm, MPI_DOUBLE, down_nbr, 0, ring_comm, &r[1] );
   MPI_Irecv( xlocal + maxm * (lrow + 1), maxm, MPI_DOUBLE, up_nbr, 1,
              ring_comm, &r[3] );
   MPI_Isend( xlocal + maxm * lrow, maxm, MPI_DOUBLE, up_nbr, 0,
              ring_comm, &r[0] );
   /* Send down, then receive from above */
   MPI_Isend( xlocal + maxm, maxm, MPI_DOUBLE, down_nbr, 1, ring_comm,
              &r[2] );
   MPI_Waitall( 4, r, statuses );
}
```

Ensure recvs posted before sends

- Irecv, Sendrecv/Barrier, Rsend, Waitall

```
void ExchangeInit( mesh )
Mesh *mesh;
{
   MPI_Irecv( xlocal, maxm, MPI_DOUBLE, down_nbr, 0, ring_comm,
              &mesh->rq[0] );
   MPI_Irecv( xlocal + maxm * (lrow + 1), maxm, MPI_DOUBLE, up_nbr, 1,
             ring_comm, &mesh->rq[1] );
}

void Exchange( mesh )
Mesh *mesh;
{
   MPI_Status statuses[2];

   /* Send up and down, then receive */
   MPI_Rsend( xlocal + maxm * lrow, maxm, MPI_DOUBLE, up_nbr, 0,
              ring_comm );
   MPI_Rsend( xlocal + maxm, maxm, MPI_DOUBLE, down_nbr, 1, ring_comm );

   MPI_Waitall( 2, mesh->rq, statuses );
}

void ExchangeEnd( mesh )
Mesh *mesh;
{
   MPI_Cancel( &mesh->rq[0] );
   MPI_Cancel( &mesh->rq[1] );
}
```

Ordered (no overlap)

- Send, Recv or Recv, Send
- MPI_Sendrecv (shift)
- MPI_Sendrecv (exchange)

Shift with MPI_Sendrecv

```
void Exchange( mesh )
Mesh *mesh;
{
    MPI_Status status;

    /* Send up, then receive from below */
    MPI_Sendrecv( xlocal + maxm * lrow, maxm, MPI_DOUBLE, up_nbr, 0,
                  xlocal, maxm, MPI_DOUBLE, down_nbr, 0, ring_comm, &status );
    /* Send down, then receive from above */
    MPI_Sendrecv( xlocal + maxm, maxm, MPI_DOUBLE, down_nbr, 1,
                  xlocal + maxm * (lrow + 1), maxm, MPI_DOUBLE, up_nbr, 1,
                  ring_comm, &status );
}
```

Use of Ssend versions

- Ssend allows send to wait until receive ready

```
void Exchange( mesh )
Mesh *mesh;
{
   MPI_Status  status;

   /* Send up, then receive from below */
   MPI_Irecv( xlocal, maxm, MPI_DOUBLE, down_nbr, 0, ring_comm, &rq );
   MPI_Ssend( xlocal + maxm * lrow, maxm, MPI_DOUBLE, up_nbr, 0,
              ring_comm );
   MPI_Wait( &rq, &status );
   /* Send down, then receive from above */
   MPI_Irecv( xlocal + maxm * (lrow + 1), maxm, MPI_DOUBLE, up_nbr, 1,
              ring_comm, &rq );
   MPI_Ssend( xlocal + maxm, maxm, MPI_DOUBLE, down_nbr, 1, ring_comm );
   MPI_Wait( &rq, &status );
}
```

Nonblocking Operations, Overlap Effective

- Isend, Irecv, Waitall
- A variant uses Waitsome with computation

```
void ExchangeStart( mesh )
Mesh *mesh;
{
    /* Send up, then receive from below */
   MPI_Irecv( xlocal, maxm, MPI_DOUBLE, down_nbr, 0, ring_comm,
              &mesh->rq[0] );
   MPI_Irecv( xlocal + maxm * (lrow + 1), maxm, MPI_DOUBLE, up_nbr, 1,
             ring_comm, &mesh->rq[1] );
   MPI_Isend( xlocal + maxm * lrow, maxm, MPI_DOUBLE, up_nbr, 0,
              ring_comm, &mesh->rq[2] );
   MPI_Isend( xlocal + maxm, maxm, MPI_DOUBLE, down_nbr, 1, ring_comm,
              &mesh->rq[3] );
}

void ExchangeEnd( mesh )
Mesh *mesh;
{
   MPI_Status statuses[4];
   MPI_Waitall( 4, mesh->rq, statuses );
}
```

Persistent Operations

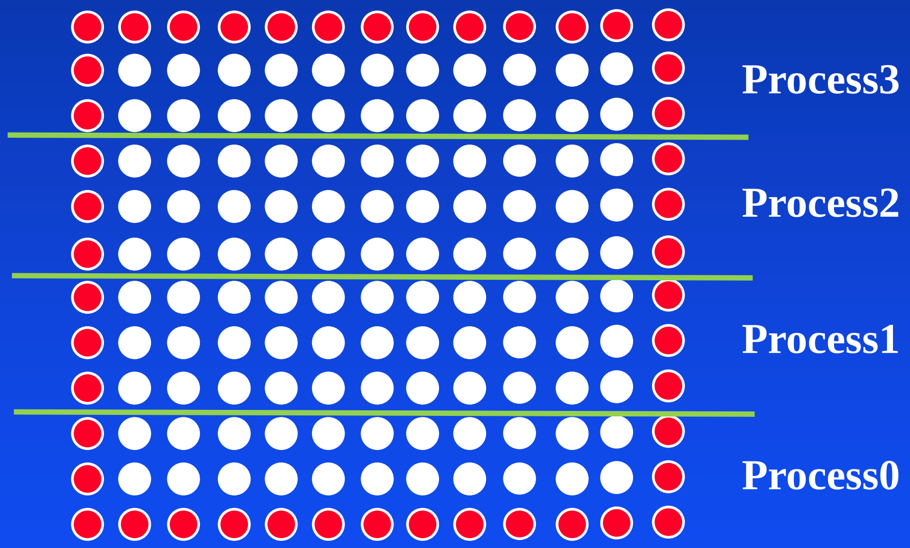
- Potential saving
  - » Allocation of MPI_Request
  - » Validating and storing arguments
- Variations of example
  - » sendinit, recvinit, startall, waitall
  - » startall(recvs), sendrecv/barrier, startall(rsends), waitall
- Some vendor implementations are buggy
- Persistent operations may be slightly *slower*

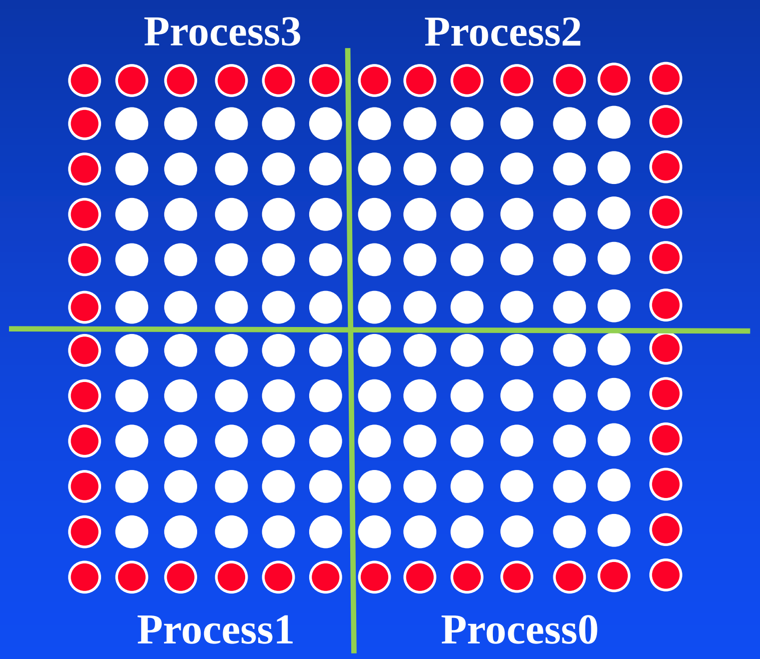
Manually Advance Automaton

- irecv, isend, iprobe in computation, waitall
- To test for messages:

```
MPI_Iprobe( MPI_ANY_SOURCE, 0,
  MPI_COMM_WORLD, &flag, &status )
```

Which Partition is better?





Communication and Computation Overlapping