Lecture 10
Parallel Algorithms III

Dr. Wilson Rivera

ICOM 6025: High Performance Computing
Electrical and Computer Engineering Department
University of Puerto Rico

Outline

- HPC Linear Algebra
- Open-MP SIMD Directives

Broadcast

Spanning Tree Algorithm

	$t = 1$	$t = 2$	$t = 3$
p_0	$x_0\downarrow, x_1\downarrow, x_2\downarrow, x_3\downarrow$	$x_0\downarrow, x_1\downarrow, x_2\downarrow, x_3\downarrow$	x_0, x_1, x_2, x_3
p_1		$x_0\downarrow, x_1\downarrow, x_2\downarrow, x_3\downarrow$	x_0, x_1, x_2, x_3
p_2			x_0, x_1, x_2, x_3
p_3			x_0, x_1, x_2, x_3

$$\lceil \log_2 p \rceil \alpha + n\beta.$$

Broadcast

scatter and a bucket brigade algorithm

	$t = 0$	$t = 1$	*etcetera*
p_0	$x_0 \downarrow$	$x_0 \qquad x_3 \downarrow$	$x_0, \quad x_2, x_3$
p_1	$x_1 \downarrow$	$x_0 \downarrow, x_1$	$x_0, x_1, \quad x_3$
p_2	$x_2 \downarrow$	$x_1 \downarrow, x_2$	x_0, x_1, x_2
p_3	$x_3 \downarrow$	$x_2 \downarrow, x_3$	x_1, x_2, x_3

$$p\alpha + \beta n(p-1)/p$$

Reduction

	$t = 1$	$t = 2$	$t = 3$
p_0	$x_0^{(0)}, x_1^{(0)}, x_2^{(0)}, x_3^{(0)}$	$x_0^{(0:1)}, x_1^{(0:1)}, x_2^{(0:1)}, x_3^{(0:1)}$	$x_0^{(0:3)}, x_1^{(0:3)}, x_2^{(0:3)}, x_3^{(0:3)}$
p_1	$x_0^{(1)} \uparrow, x_1^{(1)} \uparrow, x_2^{(1)} \uparrow, x_3^{(1)} \uparrow$		
p_2	$x_0^{(2)}, x_1^{(2)}, x_2^{(2)}, x_3^{(2)}$	$x_0^{(2:3)} \uparrow, x_1^{(2:3)} \uparrow, x_2^{(2:3)} \uparrow, x_3^{(2:3)} \uparrow$	
p_3	$x_0^{(3)} \uparrow, x_1^{(3)} \uparrow, x_2^{(3)} \uparrow, x_3^{(3)} \uparrow$		

$$\lceil \log_2 p \rceil (\alpha + n\beta + \frac{p-1}{p}\gamma n).$$

Allgather/Allreduce

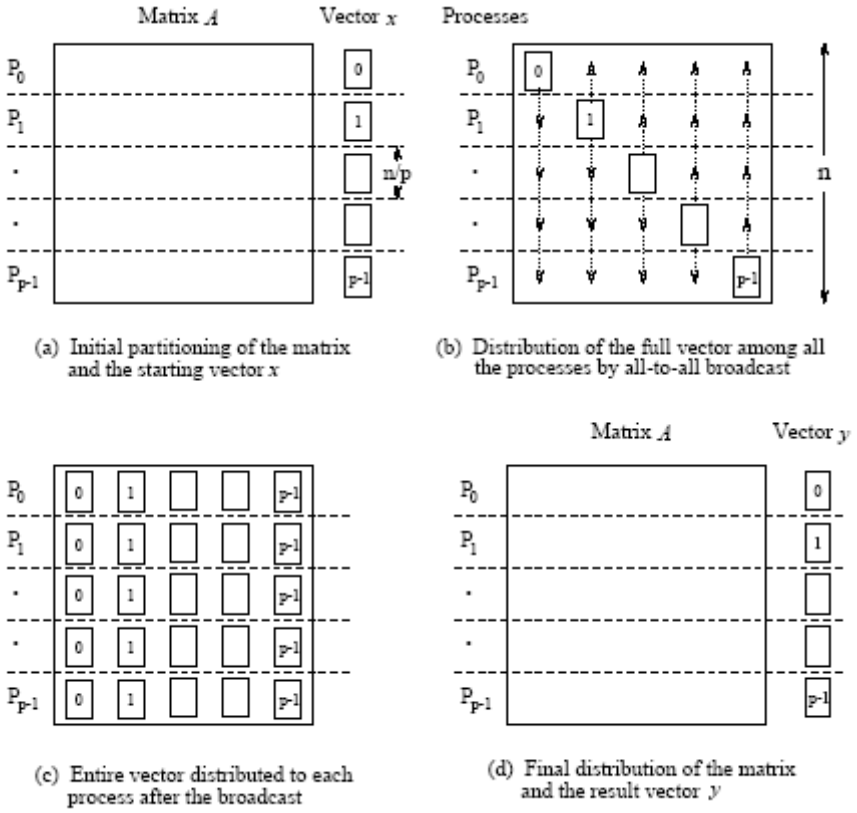
Allgather

$$\lceil \log_2 p \rceil \alpha + \frac{p-1}{p} n\beta.$$

Allreduce

$$\lceil \log_2 p \rceil \alpha + 2\frac{p-1}{p} n\beta + \frac{p-1}{p} n\gamma.$$

Matrix-Vector Multiplication



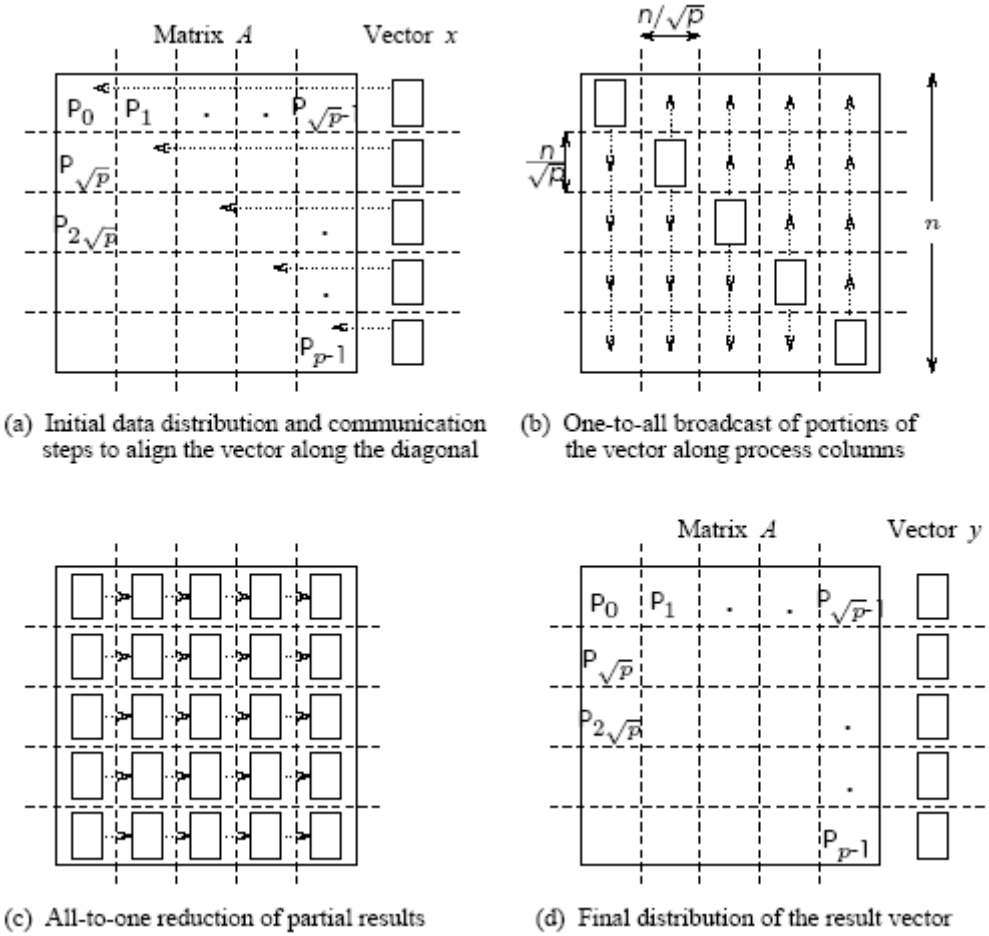
Rowwise 1-D Partitioning

Matrix-Vector Multiplication:

$$T_P = \frac{n^2}{p} + t_s \log p + t_w n.$$

Overall iso-efficiency is $W = O(p^2)$.

Matrix-Vector Multiplication



(a) Initial data distribution and communication steps to align the vector along the diagonal

(b) One-to-all broadcast of portions of the vector along process columns

(c) All-to-one reduction of partial results

(d) Final distribution of the result vector

2-D Partitioning

Matrix-Vector Multiplication: 2-D Partitioning

$$T_P \approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p$$

$$W = n^2 = p \log^2 p$$

$$p = O\left(\frac{n^2}{\log^2 n}\right)$$

Matrix-Matrix Multiplication

- The two broadcasts take time

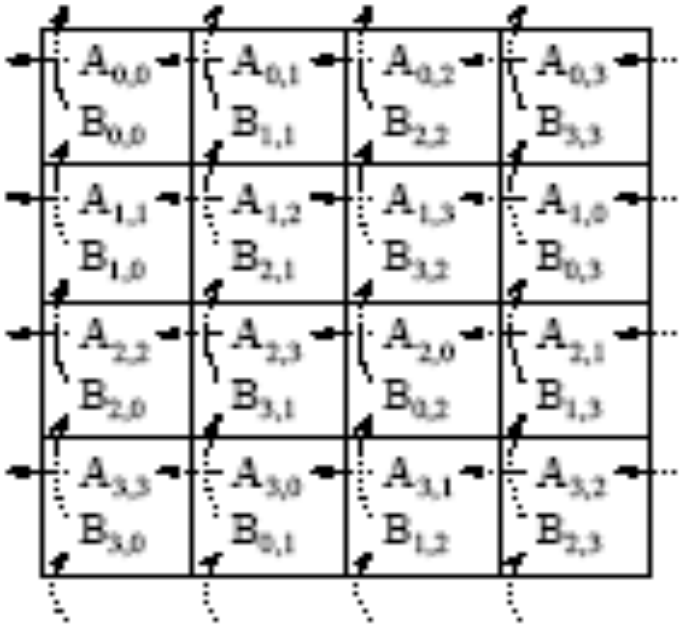
$$2(t_s \log(\sqrt{p}) + t_w(n^2/p)(\sqrt{p} - 1))$$

- The computation requires $\sqrt{p}$ multiplications of $(n/\sqrt{p}) \times (n/\sqrt{p})$ sized sub-matrices.
- The parallel run time is approximately

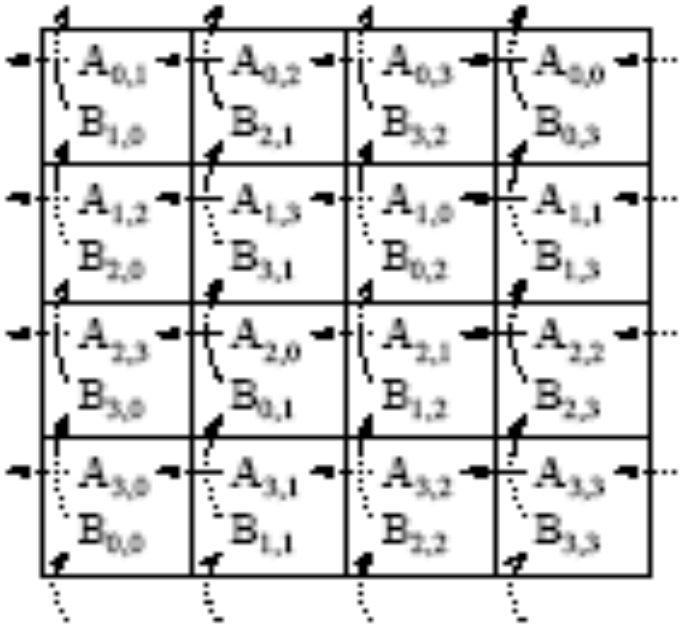
$$T_P = \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}.$$

- The algorithm is cost optimal and the isoefficiency is $O(p^{1.5})$ due to bandwidth term t_w and concurrency.
- Major drawback of the algorithm is that it is not memory optimal.

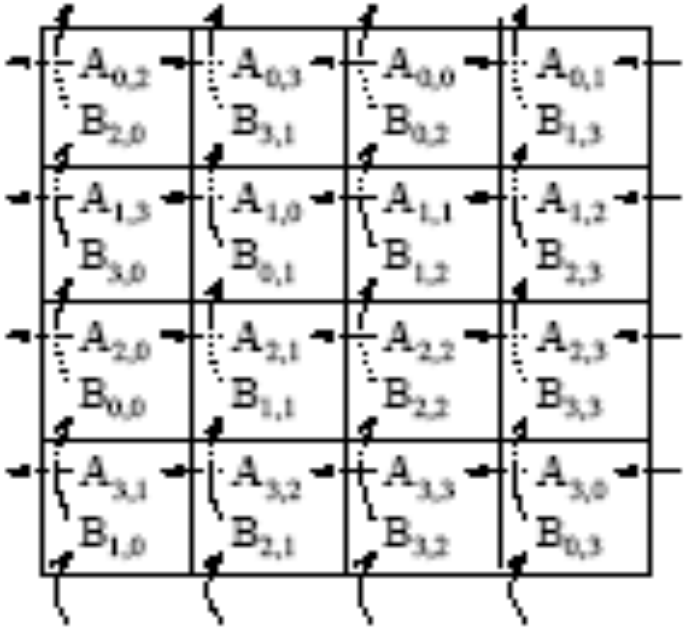
Matri
Cann



(c) A and B after initial alignment



(d) Submatrix locations after first shift



(e) Submatrix locations after second shift

(f) Submatrix locations after third shift

Matrix-Matrix Multiplication: Cannon's Algorithm

$$T_P = \frac{n^3}{p} + 2\sqrt{p}t_s + 2t_w\frac{n^2}{\sqrt{p}}.$$

The algorithm is cost optimal and the iso-efficiency is $O(p^{1.5})$

Memory efficient

Open-MP SIMD Directives

- Open-MP 4.0 offers a portable alternative for vector programming
- Multiple iterations are executed simultaneously through SIMD instructions

Open-MP SIMD Directives

- loops must be perfectly nested (there must be no code nor any Open-MP directive between any two loops).
- The loop body must be a structured block (an executable statement, possibly compound, with a single entry and exit points at the top and bottom respectively).
- The loop iteration number must be known before entering the loop.

Open-MP SIMD Directives

- safelen(length) specifies the number of iterations that can be executed simultaneously.
- linear(*list:[linear-step]*) indicates that, on each iteration, all the values of the variables included in the list correspond to their initial value before entering the construct plus the logical number of the iteration times *linear-step*.
- aligned(*list:[alignment]*) declares that the memory addresses of the variables included in the list are aligned to the number of bytes specified in *alignment*.

Open-MP SIMD Directives

- simdlen(*length*) generates a function to support a given vector length.
- uniform(*argument-list*) specifies that the arguments included in the list have a constant value between calls from the same loop execution.
  - An argument can not be in both uniform and linear clauses.
- inbranch indicates that the function is always called from inside a conditional statement.
- notinbranch indicates that function is never called from inside a conditional statement.

Example: Matrix Multiplication

```
# pragma omp for
for ( i = 0; i < n; i++ ){
  for ( j = 0; j < n; j++ ){
    c[i*n+j] = 0.0;
    for ( k = 0; k < n; k++ ){
      c[i*n+j] = c[i*n+j] + a[i*n+k] * b[k*n+j];
}}}
```

Accesses to the B array are to non-contiguous memory locations, which decreases the efficiency of vectorization as non-one-strided memory accesses are difficult to vectorize.

Ponte C. et. al. (2017). "Evaluation of OpenMP SIMD Directives on Xeon Phi Coprocessors. International Conference on High Performance Computing & Simulation

Example: Matrix Multiplication

- Avoid non-sequential accesses
  - Transpose matrix B
  - adapt the memory accesses so that the same operations are performed.
- Add SIMD directives
- Memory addresses are aligned to the VPU width (ALIGNMENT variable).
- Note that multidimensional arrays need to be padded

Example: Matrix Multiplication

```
# pragma omp for
for (i=0; i<n; i++){
for (j=0; j<n; j++){
trans[i*npadded+j] = b[j*npadded+i];
}}

# pragma omp for
for ( i = 0; i < n; i++ ){
for ( j = 0; j < n; j++ ){
temp = 0;
# pragma omp simd aligned(a,trans:ALIGNMENT) reduction(+:temp)
for ( k = 0; k < n; k++ ){
temp += a[i*npadded+k] * trans[j*npadded+k];
}
c[i*npadded+j] = temp;
}}
```

https://github.com/christianponte/omp4simd.

Example: Poisson Eq.

```
void sweep ( int nx, int ny, ... ){
  [...]
  # pragma omp parallel shared(dx,dy,f,itnew,itold,nx,ny,u,unew) private(i,it,j)
  for ( it = itold + 1; it <= itnew; it++ ){
    // Save step
    # pragma omp for
    for ( i = 0; i < nx; i++ ){
      for ( j = 0; j < ny; j++ ){
        u[i*ny+j] = unew[i*ny+j];
      }
    }
    // Compute step
    # pragma omp for
    for ( i = 0; i < nx; i++ ){
      for ( j = 0; j < ny; j++ ){
        if ( i == 0 || j == 0 || i == nx - 1 || j == ny - 1 ){
          unew[i*ny+j] = f[i*ny+j];
        } else {
          unew[i*ny+j] = 0.25 * ( u[(i-1)*ny+j] + u[i*ny+j+1] + u[i*ny+j-1] +
          u[(i+1)*ny+j] + f[i*ny+j] * dx * dy );
  }}}}
  return;
}
```

Example: Poisson Eq.

```
void sweep ( int nx, int ny, ... ){
  [...]
  # pragma omp parallel shared(dx,dy,f,itnew,itold,nx,ny,u,unew) private(i,it,j)
  for ( it = itold + 1; it <= itnew; it++ ){
    # pragma omp for
    for ( i = 0; i < nx; i++ ){
      # pragma omp simd aligned(u:ALIGNMENT, unew:ALIGNMENT)
      for ( j = 0; j < ny; j++ ){
        u[i*npadded+j] = unew[i*npadded+j];
      }
    }
    # pragma omp simd aligned(unew:ALIGNMENT, f:ALIGNMENT)
    for (i=0; i<ny; i++){
      unew[i] = f[i];
    }
    # pragma omp simd aligned(unew:ALIGNMENT, f:ALIGNMENT)
    for (i=0; i<ny; i++){
      unew[(nx-1)*npadded + i] = f[(nx-1)*npadded + i];
    }
    # pragma omp for
    for ( i = 1; i < nx-1; i++ ){
      unew[i*npadded] = f[i*npadded];
      # pragma omp simd aligned(unew:ALIGNMENT, u:ALIGNMENT, f:ALIGNMENT)
      for ( j = 1; j < ny-1; j++ ){
        unew[i*npadded+j] = 0.25 * ( u[(i-1)*npadded+j] + u[i*npadded+j+1] +
        u[i*npadded+j-1] + u[(i+1)*npadded+j] + f[i*npadded+j] * dx * dy );
      }
      unew[i*npadded + ny-1] = f[i*npadded + ny-1];
  }}
  return;
}
```

Example: Molecular Dynamics

```
void compute ( int np, int nd, double pos[], ... ){
[...]
# pragma omp parallel shared ( f, nd, np, pos, vel ) private ( i, j, k, rij, d, d2 )
# pragma omp for reduction ( + : pe, ke )
for ( k = 0; k < np; k++ ){
  for ( i = 0; i < nd; i++ ){
    f[i+k*nd] = 0.0;
  }
  for ( j = 0; j < np; j++ ){
    if ( k != j ){
      d = dist ( nd, pos+k*nd, pos+j*nd, rij );
      if ( d < PI2 ){
        d2 = d;
      } else {
        d2 = PI2;
      }
      pe = pe + 0.5 * pow ( sin ( d2 ), 2 );
      for ( i = 0; i < nd; i++ ){
        f[i+k*nd] = f[i+k*nd] - rij[i] * sin ( 2.0 * d2 ) / d;
  }}}
  for ( i = 0; i < nd; i++ ){
    ke = ke + vel[i+k*nd] * vel[i+k*nd];
  }
}
[...]
}
```

Example: Molecular Dynamics

```
void compute ( int np, int nd, const int nppadded, double pos[], ... ){
  [...]
  # pragma omp parallel shared ( f, nd, np, pos, vel ) private ( i, j, k, rij, d, d2 )
  {
    [...]
    # pragma omp for reduction ( + : pe, ke )
    for ( k = 0; k < np; k++ ){
      # pragma omp simd reduction(+:pe) aligned(d,pos,rij,d2:ALIGNMENT)
      for (j=0; j<np; j++){
        d[j] = pt_dist(nd, nppadded, pos, k, j, rij);
        d2[j] = min(d, j, PI2);
        pe += 0.5 * pow(sin(d2[j]), 2);
      }
      # pragma omp simd reduction(+:ke) aligned(f,vel:ALIGNMENT)
      for ( i = 0; i < nd; i++ ){
        f[i*nppadded+k] = 0.0;
        ke += vel[i*nppadded+k] * vel[i*nppadded+k];
      }
      for ( i = 0; i < nd; i++ ){
        # pragma omp simd aligned(f,rij,d,d2:ALIGNMENT)
        for ( j = 0; j < np; j++ ){
          f[i*nppadded+k] -= f_calc(nppadded, rij, d, d2, i, j, k);
    }}}
    [...]
  }
  [...]
}
```

Performance Results