The MathWorks

MATLAB® & SIMULINK®

Parallel MATLAB

HPCC Class 2 Submission for SC|08

Jos Martin

jos.martin@mathworks.co.uk

Parallel MATLAB

- Language, Libraries and Environment

- Interactive
- Scalable
  - Prototyping from desktop to large cluster
  - Language constructs
  - Algorithms
- Mix serial and parallel programming models

Parallel Language Constructs

- `parfor ... end`
  - Parallel for loop
  - Loop contains independent iterates or reduction operations
- `spmd ... end`
  - Single Program Multiple Data programming model
  - `labindex` and `numlabs` setup correctly
  - Inter-lab communication library becomes useful
  - Distributed Array library builds on this infrastructure

Distributed Arrays

- Currently 2 distribution schemes
  - 1D generalized partition in any dimension
  - 2D block cyclic
- Redistribution within and between schemes
- On 1D
  - Most MATLAB mathematical functions supported
  - Most MATLAB indexing supported
- On 2D
  - ScaLAPACK functionality

Environment

- `matlabpool open`
  - Communication with arbitrary scheduler
  - Startup MATLAB backend processes
  - Communicate with each other (MPI) and the client

HPL

```
function HPC_linpack(n)
spmd
    % Create a distributed matrix and a replicated vector
    A = rand(n, n, codistributor);
    A = redistribute(A, codistributor('2d'));
    b = rand(n, 1, codistributor);
    % Time the solution of the linear system
    tic; % Start timing
    x = A\b;
    t = toc; % Stop timing
    % Calculate scaled residuals
    A = redistribute(A, codistributor('1d'));
    r1 = norm(A*x-b,inf)/(eps*norm(A,1)*n);
    r2 = norm(A*x-b,inf)/(eps*norm(A,1)*norm(x,1));
    r3 = norm(A*x-b,inf)/(eps*norm(A,inf)*norm(x,inf)*n);
    if max([r1 r2 r3]) > 16
        error('Failed the HPC HPL Benchmark');
    end
end
% Performance in gigaflops
perf = (2/3*n^3 + 3/2*n^2)/max([t{:}])/1.e9;
fprintf('Data size: %f GB\nPerformance: %f GFlops\n', 8*n^2/(1024^3), perf);
```

Make Distributed Arrays

Timed HPL region

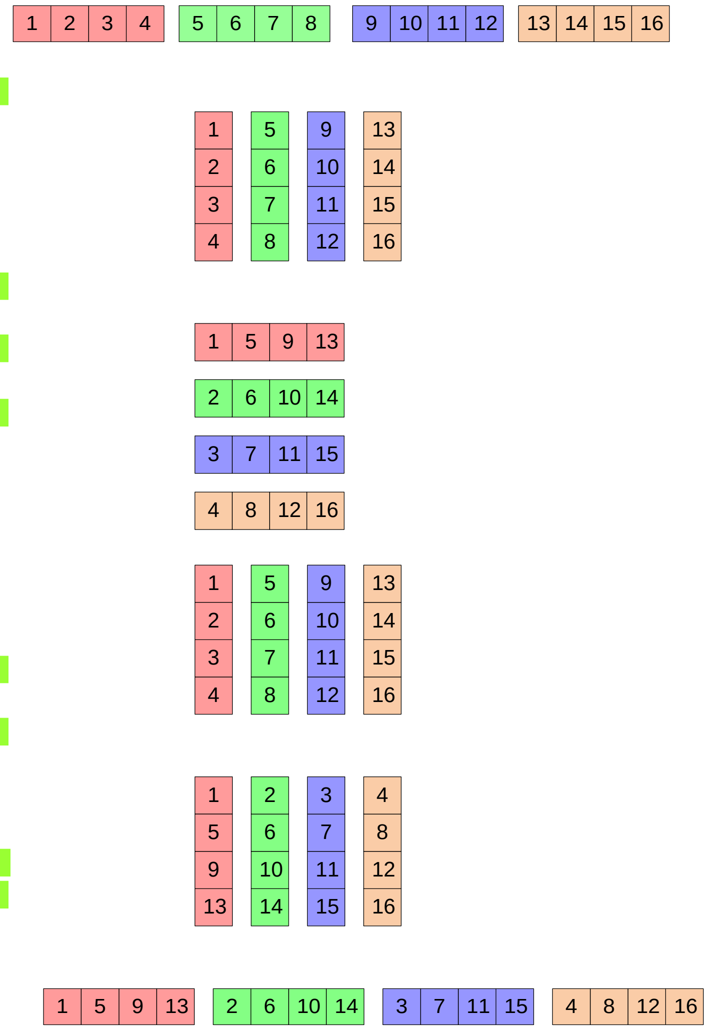
Check result is within tolerance

FFT

```
function HPC_fft(m)
spmd
    % Create complex 1 x m random vector
    x = complex( rand(1, m, codistributor), rand(1, m, codistributor) );
    % Time the forward FFT
    tic; % Start timing
    y = dfft(x);
    t = toc; % Stop timing
    % Performance in gigaflops
    perf = 5*m*log2(m)/t/1.e9;
    % Compute error from the inverse FFT
    z = (1/length(y))*conj(dfft(conj(y)));
    err = norm(x-z,inf)/(16*log2(m)*eps);
end
perf = min([perf{:}]);
err = err{1};
if err > 1
    error('Failed the HPC FFT Benchmark');
end
fprintf('Data size: %f GB\nPerformance: %f GFlops\nErr: %f\n', ...
    32*m/(1024^3), perf, err);
```

FFT

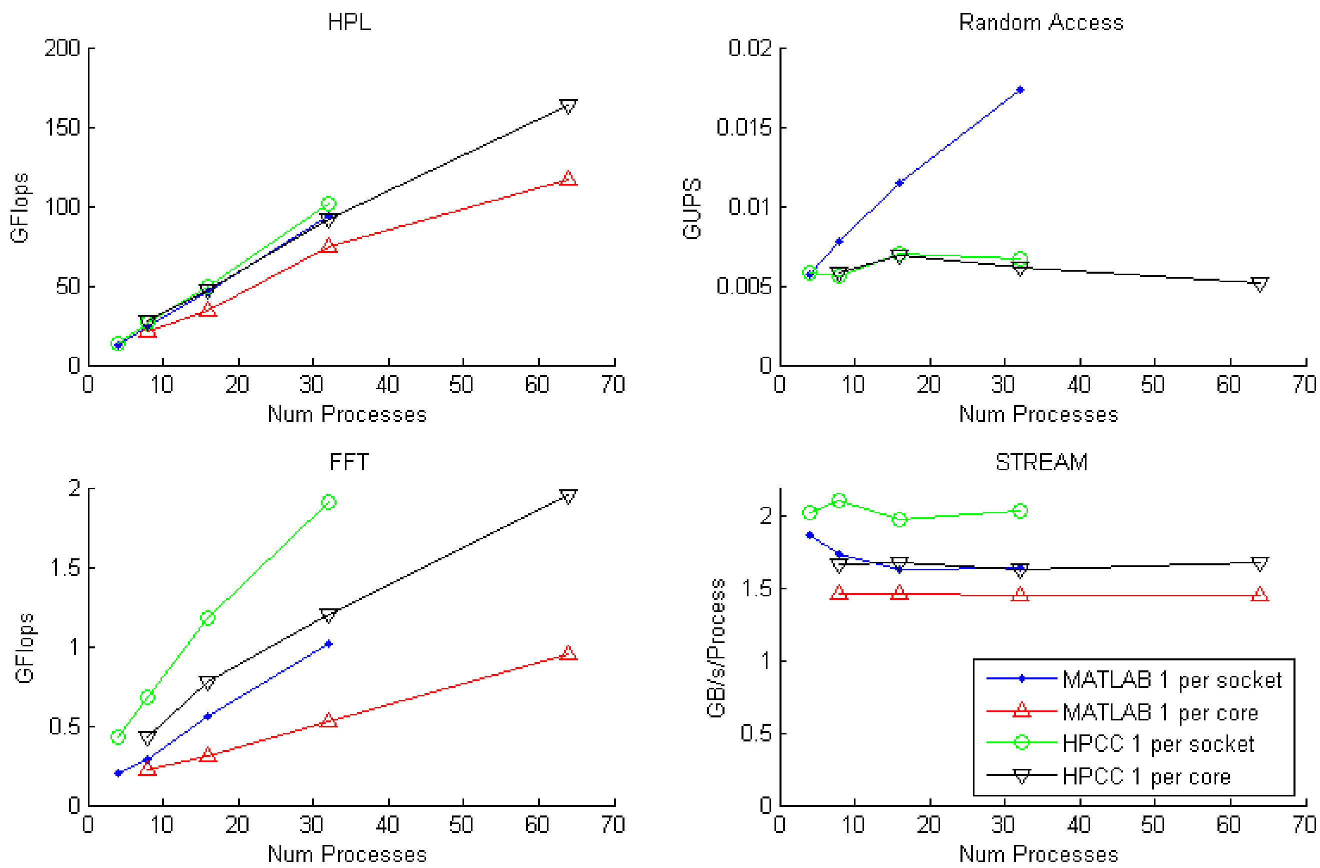
```
function x = dfft(x)
% Remember row or column size
s = size(x);
% Reshape to matrix with numlabs columns
M = numlabs;
N = prod(s)/M;
x = iReshape(x, N, M);
% Redistribute to do small FFT's on each lab
x = redistribute(x, codistributor('1d', 1));
% Local 1-D FFT in 2nd dimension
xloc = fft(localPart(x), [], 2);
% Compute local twiddle factors
omega = exp(-2*pi*1i*(localPart(codcolon(0,N-1))')/prod(s));
t = repmat(omega, 1, M); t(:, 1) = 1;
t = cumprod(t, 2);
% Multiply by the local twiddle factors
x = codistributed(xloc .* t, codistributor(x));
% Redistribute to do second set of small FFT's on each lab
x = redistribute(x, codistributor('1d', 2));
% Local 1-D FFTs in 1st dimension
xloc = fft(localPart(x), [], 1);
% Recreate distributed array
x = codistributed(xloc,  codistributor(x));
% Return distributed array row or column vector
x = redistribute(x.', codistributor('1d', 2));
x = iReshape(x, s(1), s(2));

function B = iReshape(A, r, c)
L = codistributed(reshape(localPart(A), r, c/numlabs));
```



Random Access

```
% Skip to position in random stream - C implementation
RArand( (labindex-1) * m * 4 / numlabs, 'StreamOffset' );
tic; % Start timing
for k = 1:nloops
    % C RNG - lastR = (lastR << 1)^(lastR & BIT64 ? POLY : ZERO64B);
    list = RArand( blockSize );
    for d = 0 : log2Numlabs-1
        % Choose partner
        partner       = 1 + bitxor( (labindex-1), 2.^d );
        % Choose mask for this dimension of the hypercube
        dim_mask      = uint64( 2.^( d + log2LocalSize ) );
        % Choose data to send and receive for this dimension
        list_and_mask = logical( bitand( list, dim_mask ) );
        if partner > labindex
            should_send = list_and_mask;
        else
            should_send = ~list_and_mask;
        end
        send_list = list( should_send );
        keep_list = list( ~should_send );
        % Use send/receive to get some data that we should keep
        recv_list = labSendReceive( partner, partner, send_list );
        % Our new list is the old list and what we've received
        list      = [keep_list, recv_list];
    end
    % Finally, after all communication, perform the  table updates.
    idx = 1 + double( bitand( localMask, list ) );
    T(idx) = bitxor( T(idx), list );
end
% Calculate max time
t = gop( @max, toc ); % Stop timing
```

Performance



HPCC Source Code Lines

	Timed Region	Total
`Stream`	1	19
`HPL`	1	18
`FFT`	22	40
`RandomAccess`	20 (m) + 10 (C)	53(m) + 35 (C)

Interested?

- Come and visit us at booth 1841