An Interactive Approach to Parallel Combinatorial Algorithms with Star-P

Alan Edelman MIT (& ISC)
John Gilbert & Viral Shah UCSB
Steve Reinhardt & Todd Letsche SGI
Parallel Computing Arts

Message Passing: The King’s Messenger

Batch Processing: Coding, Modeling, & Debugging

Punch Cards (textile loom 1840)

Noble perfected arts: what’s next for productivity?
Productivity

Make this machine go faster?

Most important catalysts for productivity are

Interactivity & ease of use

puzzle pieces working together

Humans interacting online
Star-P = A Software Platform For Interactive Supercomputing

“The Dream”
INTERACTIVE Fundamentally Alters the Flawed Process

Re-coding takes time, and invariably takes away from model refinement

Model Development Phase – “Time to Production”

Batch Workflow

Desktop Prototyping

Transfer to HPC (re-code in C/Ftn, MPI)

Test and Scale Model With Real Data

Production

Limited iterations

Interactive Workflow

“No change in religion”

“No change in religion”

“Time to Production”

Interactive Time Savings

“ISC is the closest thing I’ve seen to a killer app.”

John Mucci
CEO, SiCortex
High Productivity Design Principles

1. **Rich set of High Performance primitives & tools.**
   a. Interoperate
   b. Interactive

2. **OK to exploit special-purpose hardware as appropriate (FGPGAs, GPUs)**

3. **Do it yourself (in MPI, OpenMP, etc.) → do it for everyone!**
The Buffon Needle Problem

\[ P(l; a, b) = \frac{(2l(a+b) - l^2)}{(\pi ab)} \]

```matlab
function z = buffon(a, b, l, trials)
    r = rand(trials, 3);
    x = a*r(:,1) + l*cos(2*pi*r(:,3)); % x coord
    y = b*r(:,2) + l*sin(2*pi*r(:,3)); % y coord
    inside = (x >= 0) & (y >= 0) & (x <= a) & (y <= b);

    bpi = (2*l*(a+b) - l^2) / (a*b*(1-sum(inside)/trials));

    z = [buffonpi; pi; abs(pi - buffonpi)/pi];
end
```

`buffon(1,1,.5,10000*p)`
Star-P Language

1. MATLAB™, plus
2. Global view (ν. node-oriented)
3. Strong bias towards propagation of distributed attribute
4. *p denotes dimension of distributed array
5. Overloading of operators
6. ppeval for task parallelism
7. Empirical data: typically have to change 10-20 SLOC for MATLAB codes to work in Star-P

```plaintext
xxx == explicit parallel extension
yyy == parallelism propagated implicitly

a = rand(n,n*p);
ppload imagedata a

[nrow ncol] = size(a);
b = ones(nrow,ncol);
c = fft2(a);
d = ifft2(c);

diff = max(max(abs(a-d)));
if (diff > 10*eps)
    sprintf('Error, diff=%f', diff);
end

e = ppeval('sum',a);
e = ppeval('quad', 'fun', a);
```
Combinatorial Algorithm Design Principle:
Do it with a sparse matrix

Graph Operations are well expressed with sparse matrices as the data structure.

Primitives for combinatorial scientific computing.

a. Random-access indexing: \( A(i,j) \)
b. Neighbor sequencing: \( \text{find} \ (A(i,:)) \)
c. Sparse table construction: \( \text{sparse} \ (I, J, V) \)
d. Matrix * Vector: walking on the graph
Star-P sparse data structure

• **Full:**
  - 2-dimensional array of real or complex numbers
  - \((nrows*ncols)\) memory

• **Sparse:**
  - compressed row storage
  - about \((2*nzs + nrows)\) memory
Star-P distributed sparse data structure

Each processor stores:
- # of local nonzeros
- range of local rows
Scalable Synthetic Compact Application (SSCA) Benchmarks

1. Bioinformatics Optimal Pattern Matching
2. Graph Theory
3. Sensor Processing

SSCA#2:- Graph Analysis; stresses memory access; compute-intensive and hard to parallelize.
Kernel 1: Construct graph data structures
  Bulk of time for smaller problems
Kernel 2: Search within large sets
Kernel 3: Subgraph extraction
Kernel 4: Graph clustering
  Version does not scale for larger problems

OpenMP Contest:
http://www.openmp.org/drupal/sc05/omp-contest.htm
1. First prize: $1000 plus a 60GB iPod.
2. Second prize: $500 plus a 4GB iPod nano.
3. Third prize: $250 plus a 1GB iPod shuffle
Scalability

Kernels 1 through 3 ran on $N=2^{26}$

- Previous largest known run is $N=2^{21}$ or 32 times smaller on a Cray MTA-2

- Timings scale reasonably – we played with building the largest sparse matrix we could, until we hit machine limitations!
  - 2xProblem Size $\rightarrow$ 2xTime
  - 2xProblem Size & 2xProcessor Size $\rightarrow$ same time
## Lines of Code

Lines of executable code (excluding I/O and graphics based on original codes available):

<table>
<thead>
<tr>
<th></th>
<th>cSSCA2</th>
<th>The spec</th>
<th>Pthreads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel 1</td>
<td>29</td>
<td>68</td>
<td>256</td>
</tr>
<tr>
<td>Kernel 2</td>
<td>12</td>
<td>44</td>
<td>121</td>
</tr>
<tr>
<td>Kernel 3</td>
<td>25</td>
<td>91</td>
<td>297</td>
</tr>
<tr>
<td>Kernel 4</td>
<td>44</td>
<td>295</td>
<td>241</td>
</tr>
</tbody>
</table>
### Expressive Power: SSCA#2 Kernel 3

**Star-P (25 SLOC)**

```matlab
A = spones(G.edgeWeights{1});
nv = max(size(A));
npar = length(G.edgeWeights);
nstarts = length(starts);
for i = 1:nstarts
    v = starts(i);
    x = zeros(nv,1); x(v) = 1;
    for k = 1:pathlen
        x = A*x;
        x = (x ~= 0);
    end;
vtxmap = find(x);
S.edgeWeights{1} = G.edgeWeights{1}...
    (vtxmap,vtxmap);
    for j = 2:npar
        sg = G.edgeWeights{j}(vtxmap,vtxmap);
        if nnz(sg) == 0
            break;
        end;
        S.edgeWeights{j} = sg;
    end;
S.vtxmap = vtxmap;
subgraphs{i} = S;
end
```

**MATLABmpi (91 SLOC)**

```matlab
declear_globals;
intSubgraphs = subgraphs(G, pathLength, startSetInt);
strSubgraphs = subgraphs(G, pathLength, startSetStr);
%| Finish helping other processors.
if P.Ncpus > 1
    if P.myRank == 0 % if we are the leader
        for unused = 1:P.Ncpus-1
            [src tag] = probeSubgraphs(G, [P.tag.K3.results]);
            [isg ssg] = MPI_Recv(src, tag, P.comm);
            intSubgraphs = [intSubgraphs isg];
            strSubgraphs = [strSubgraphs ssg];
        end
        for dest = 1:P.Ncpus-1
            MPI_Send(dest, P.tag.K3.done, P.comm);
        end
    else
        MPI_Send(0, P.tag.K3.results, P.comm, ...
            intSubgraphs, strSubgraphs);
        [src tag] = probeSubgraphs(G, [P.tag.K3.done]);
        MPI_Recv(src, tag, P.comm);
    end
end
```

### cSSCA2 executable spec C/Pthreads SIMPLE

<table>
<thead>
<tr>
<th>Kernel</th>
<th>cSSCA2</th>
<th>executable</th>
<th>C/Pthreads</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>29</td>
<td>68</td>
<td>256</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>44</td>
<td>121</td>
</tr>
<tr>
<td>3</td>
<td>25</td>
<td>91</td>
<td>297</td>
</tr>
<tr>
<td>4</td>
<td>44</td>
<td>295</td>
<td>241</td>
</tr>
</tbody>
</table>
Did not just build a benchmark: Explored an algorithm space!

Spectral Partitioning based on Parpack was fine for small sizes but not larger.

We played around! We plotted data! We had a good time. 😊 Parallel computing is fun again!
Interactive Supercomputing

1. No “change in religion”
   a. Use familiar tools
   b. Desktop, interactive

2. 5-10x manpower savings by transforming workflow
   a. Enables rapid (and more frequent) iteration
   b. Drives better conclusions, decisions, products

3. Improves “Time to Production”
   a. 50% reductions in calendar time
   b. Improves time to market
   c. Increases profits

“"In computing with humans, response time is everything....One's likelihood of getting the science right falls quickly as one loses the ability to steer the computation on a human time scale."

Prof. Nick Trefethen
Oxford University
Contact: Alan Edelman

Interactive Supercomputing • 135 Beaver Street, FL 2 • Waltham, MA 02452
781.398-0010 • edelman@interactivesupercomputing.com • www.interactivesupercomputing.com