Experiments with \texttt{bsplu}

(PSC §2.5–2.6)
Broadcast function

```c
void bsp_broadcast(double *x, int n, int src,
        int s0, int stride, int p0,
        int s, int phase){
    /* Broadcast the vector x of length n
     * from processor src to processors s0+t*stride, 0 <= t < p0. x has already been registered.
     *
     * s = local processor identity.
     * phase = phase of two-phase broadcast (0 or 1)
     * Only one phase is performed, without sync. */
```

- Standard 1D–2D identification $P(s, t) \equiv P(s + tM)$.
- $\text{stride} = 1, p0 = M$: broadcast within processor column.
- $\text{stride} = M, p0 = N$: broadcast within processor row.
- No sync to allow combining supersteps.
Phase 0: source processor spreads the data

\[ b = \begin{cases} \frac{n}{p_0} & \text{if } n \% p_0 == 0 \\ \frac{n}{p_0} + 1 & \text{otherwise} \end{cases}; \quad /* \text{block size} */ \]

\[
\text{if} \ (\text{phase} == 0 \ \&\& \ s == \text{src}) \{
\quad \text{for} \ (t = 0; t < p_0; t++) \{
\quad \quad \text{dest} = s_0 + t \times \text{stride};
\quad \quad \text{nbytes} = \text{MIN}(b, n - t \times b) \times \text{SZDBL};
\quad \quad \text{if} \ (\text{nbytes} > 0)
\quad \quad \quad \text{bsp\_put}(\text{dest}, \&x[t \times b], x, t \times b \times \text{SZDBL}, \text{nbytes});
\quad \}
\}
\]

Data is put in the same location \( t \cdot b \) of array \( x \) in the destination processor as in the source processor.
Phase 1: participating processors perform broadcast

```c
if (phase==1 && s%stride==s0%stride){
    t=(s-s0)/stride; /* s = s0+t*stride */
    if (0<=t && t<p0){
        nbytes= MIN(b,n-t*b)*SZDBL;
        if (nbytes>0){
            for (t1=0; t1<p0; t1++){
                dest= s0+t1*stride;
                if (dest!=src)
                    bsp_put(dest,&x[t*b],x,
                            t*b*SZDBL,nbytes);
            } /* for */
        } /* if */
    } /* if */
} /* if */
```

Data is not sent back to the source. No influence on BSP cost, but it reduces the communication volume. This cannot be bad.
Local and global indices for cyclic distribution

<table>
<thead>
<tr>
<th>Global</th>
<th>12</th>
<th>0</th>
<th>4</th>
<th>7</th>
<th>-1</th>
<th>2</th>
<th>15</th>
<th>11</th>
<th>3</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Local</th>
<th>12</th>
<th>-1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>4</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>7</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Global index: $i$  
Local index on $P(s)$: $i$
Relation: $i = i \cdot p + s$

```c
/* Initialise permutation vector pi */
nlr= nloc(M,s,n); /* number of local rows */
if (t==0)
    for(i=0; i<nlr; i++)
        pi[i]= i*M+s; /* global row index */
```
Putting data directly into a 2D array

```c
a = matallocd(nlr, nlc); /* in bsplu_test.c */
void bsplu( ... , int *pi, double **a){
    double *pa = NULL;
    if (nlr>0)
        pa = a[0];
    bsp_push_reg(pa,nlr*nlc*SZDBL);
    bsp_push_reg(pi,nlr*SZINT);
    ...
    if (k%M==s){
        /* Store pi(k) in pi(r) on P(r%M,0) */
        if (t==0)
            bsp_put(r%M,&pi[k/M],pi,(r/M)*SZINT,SZINT);
        /* Store row k of A in row r on P(r%M,t) */
        bsp_put(r%M+t*M,a[k/M],pa,
               (r/M)*nlc*SZDBL,nlc*SZDBL);
    }
    ...}
```
Two-phase broadcast of column $k$

double *lk;

nlr = nloc(M,s,n); /* number of local rows */

kr = nloc(M,s,k); /* first local row with global index >= k */

kc = nloc(N,t,k);

kr1 = nloc(M,s,k+1);

lk = vecallocd(nlr); bsp_push_reg(lk, nlr*SZDBL);

... 

if (k%N==t) /* Store new column k in lk */
    for (i = kr1; i < nlr; i++)
        lk[i-kr1] = a[i][kc];

bsp_broadcast(lk, nlr-kr1, s+(k%N)*M, s, M, N, s+t*M, 0);

bsp_sync();

bsp_broadcast(lk, nlr-kr1, s+(k%N)*M, s, M, N, s+t*M, 1);

bsp_sync();
### Time (in s) of LU decomposition

<table>
<thead>
<tr>
<th>$n$</th>
<th>one-phase</th>
<th>two-phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000</td>
<td>1.21</td>
<td>1.33</td>
</tr>
<tr>
<td>2 000</td>
<td>7.04</td>
<td>7.25</td>
</tr>
<tr>
<td>3 000</td>
<td>21.18</td>
<td>21.46</td>
</tr>
<tr>
<td>4 000</td>
<td>47.49</td>
<td>47.51</td>
</tr>
<tr>
<td>5 000</td>
<td>89.90</td>
<td>89.71</td>
</tr>
<tr>
<td>6 000</td>
<td>153.23</td>
<td>152.79</td>
</tr>
<tr>
<td>7 000</td>
<td>239.21</td>
<td>238.25</td>
</tr>
<tr>
<td>8 000</td>
<td>355.84</td>
<td>354.29</td>
</tr>
<tr>
<td>9 000</td>
<td>501.92</td>
<td>499.74</td>
</tr>
<tr>
<td>10 000</td>
<td>689.91</td>
<td>689.56</td>
</tr>
</tbody>
</table>

Cray T3E with $p = 64$, $r = 38.0$ Mflop/s, $g = 87$, $l = 2718$ (measured by bspbench). 8 × 8 cyclic distribution.
Total broadcast time of LU decomposition

Cray T3E with $p = 64$, $r = 38.0$ Mflop/s, $g = 87$, $l = 2718$. 
Any actual savings by two-phase broadcast?

- **Not much difference** in total time between one-phase and two-phase approach.
- For $n < 4000$, with local broadcast length $< 500$, one-phase is better.
- For $n > 4000$, two-phase is better. But savings are insignificant compared to computation time. Total broadcast time is $< 5\%$ of overall time.
- **BSP analysis gives insight and explains results**, even if they are surprising/disappointing/...
- On a different machine with slower communication, such as a PC cluster, the savings will be significant. Try it!
Total measured and predicted time
Optimistic prediction is right

- BSP model predicts: row swaps, phase 0 of the broadcast, and phase 1 all take the same time. Measurements validate this.
- **Very different communication patterns**: row swaps and phase 0 are very unbalanced, phase 1 is well-balanced.
- **Pessimists are usually wrong**. The pessimistic $g$-value (for puts of single data words) is far off.
- You need to plug the right $g$-value into the BSP cost formula to obtain meaningful predictions. `bsplu` puts elements from row and column $k$ as large data packets. Therefore, we should use the optimistic $g$-value.
Profile of stages $k = 0, 1, 2$ of an LU decomposition

Cray T3E: $n = 100$, $M = 8$, $N = 1$. Obtained by bspprof.
Game: recognise the supersteps

- $M = 8$, $N = 1$: row distribution of the matrix.
- **Column broadcast** is for free.
- **Row swap** involves two processors; each time a different pair. This must be superstep 12.
- **Phase 0 of row broadcast** has 1 sender, 7 receivers. This must be superstep 13.
- **Phase 1** has 8 senders, 7 receivers, and takes about the same time (bar width) as superstep 13. This must be superstep 14.
- The wide gap between supersteps 14 and 10 is a big computation superstep. This must be the **matrix update**.
- Superstep 10 must be the exchange of local winners in the **pivot search** with 8 senders and 8 receivers. Relatively costly, because the problem size is only $n = 100$. 
Summary

- We use **global indices** in the description of an algorithm, but **local indices** in an actual program.
- We **understand** the behaviour of our program, though we may not always like it.
- Very **different communication patterns** with the **same BSP cost** take about the **same time** on an actual parallel computer, the Cray T3E.
- **Profiling** is a way of getting intimate knowledge of your program. The superstep concept makes this very easy.