Message Passing Interface (MPI-2)
(PSC Appendix C, §C.2.5–C.4)
One-sided communications in MPI-2

bsp_hppput(pid, src, dst, dst_offset, bytes, nbytes);

MPI_Put(src, src_n, src_type, pid, dst_offset, dst_n, dst_type, dst_win);

- The standard put operation in MPI-2 is the unbuffered put, equivalent to the high-performance put in BSPlib.
- Data sizes and offsets are measured in units of the basic data type, src_type for the source array and dst_type for the destination array. Both could e.g. be MPI_DOUBLE.
- The destination memory area is not given by a pointer to memory space such as an array, but by a pointer to a window object.
Windows for one-sided communications

bsp_push_reg(variable, nbytes);

MPI_Win_create(variable, nbytes, unit, info, comm, win);

bsp_pop_reg(variable);

MPI_Win_free(win);

▶ A window is a preregistered and distributed memory area, consisting of local memory on every processor of a communicator.
▶ A window is created by MPI_Win_create, equivalent to bsp_push_reg.
▶ win is the window of type MPI_Win corresponding to the array variable.
▶ The integer unit is the unit for expressing offsets; comm is the communicator of the window.
Creating a window

MPI_Win_create(variable, nbytes, unit, info, comm, win); //syntax

MPI_Win v_win;

MPI_Win_create(v, nv*SZDBL, SZDBL, MPI_INFO_NULL, MPI_COMM_WORLD, &v_win);

MPI_Win_fence(0, v_win);

- A window can be used after a call to MPI_Win_fence, which can be thought of as a synchronisation of the processors that own the window.
Fanout in mpimv

```c
for(j=0; j<ncols; j++)
    MPI_Get(&vloc[j], 1,MPI_DOUBLE,srcprocv[j], srcindv[j],1,MPI_DOUBLE,v_win);
MPI_Win_fence(0, v_win);
```

- Communications initiated before a fence are guaranteed to have been completed after the fence.
- The fence acts as a **synchronisation** at the end of a superstep.
Fanin using accumulate

```c
for(i=0; i<nrows; i++){
    /* compute psum = local partial sum of row i */
    ...

    MPI_Accumulate(psum,1,MPI_DOUBLE,
                   destprocu[i], destindu[i],
                   1,MPI_DOUBLE,MPI_SUM,u_win);
}

MPI_Win_fence(0, u_win);
```

- Accumulate is a one-sided communication.
- Instead of putting a value into the destination location, accumulate adds a value into the location, or takes a maximum, or performs another binary operation.
Comparison of BSPlib and MPI for inner product

<table>
<thead>
<tr>
<th>Program</th>
<th>$n$</th>
<th>$p$</th>
<th>BSPlib</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inner product</td>
<td>100000</td>
<td>1</td>
<td>4.3</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>4.2</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>5.9</td>
<td>1.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>9.1</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>26.8</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- Time $T_p(n)$ (in ms) of parallel program from BSPedupack and MPIedupack on $p$ processors of a Silicon Graphics Origin 3800.
- BSPlib implementation was designed for earlier machine.
- The vendor’s version of MPI is clearly well-optimised, leading to good scalability.
Comparison of BSPlib and MPI for LU and FFT

<table>
<thead>
<tr>
<th>Program</th>
<th>n</th>
<th>p</th>
<th>BSPlib</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU decomposition</td>
<td>1000</td>
<td>1</td>
<td>5408</td>
<td>6341</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>2713</td>
<td>2744</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>1590</td>
<td>1407</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>1093</td>
<td>863</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>1172</td>
<td>555</td>
</tr>
<tr>
<td>FFT</td>
<td>262144</td>
<td>1</td>
<td>154</td>
<td>189</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>111</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>87</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>41</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>27</td>
<td>19</td>
</tr>
</tbody>
</table>
Comparison of BSPlib and MPI for matrix–vector

<table>
<thead>
<tr>
<th>Program</th>
<th>$n$</th>
<th>$p$</th>
<th>BSPlib</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix–vector</td>
<td>20 000</td>
<td>1</td>
<td>3.8</td>
<td>3.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>11.4</td>
<td>2.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>14.7</td>
<td>6.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>20.8</td>
<td>8.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
<td>18.7</td>
<td>11.0</td>
</tr>
</tbody>
</table>

Test problem amorph20k too small to obtain speedup.
How to use BSP in an MPI world?

- The first, purist approach is to write our programs in BSPlib and install BSPlib ourselves if needed.
- Main advantages: ease of use; automatic enforcement of the BSP style; no deadlock.
- For shared-memory architectures, efficient implementation is available through Albert-Jan Yzelman’s MulticoreBSP for C.
- Always possible to use BSPonMPI by Wijnand Suijlen.
- BSPonMPI is a library. Linking it with your BSPlib program turns it into an MPI program. Then use mpirun ...
Second approach: the hybrid program

- The **hybrid approach** is to write a single program in BSP style, but express all communication both in MPI and BSPlib.
- The resulting single-source program can then be **compiled conditionally** (with or without a flag `-DMPITARGET`), e.g. for the FFT:
  ```c
  #ifdef MPITARGET
   mpiredistr(x,n,p,s,c0,c,rev,rho_p);
  #else
   bspredistr(x,n,p,s,c0,c,rev,rho_p);
  #endif
  ```
- **Main advantages:** single-source program; choice of BSP or MPI, whichever is fastest; encourages programming in BSP style also in the MPI part of programs.
- **Disadvantage:** longer program texts.
Third approach: write in BSPlib, then convert to MPI

- Main advantages: saves human time when developing the program; single-source program.
- Disadvantage: some extra effort needed at the end of the development stage.
- This approach was taken for BSPedupack, which was converted into MPIedupack within a week.
Fourth approach: write in MPI-2

- Use collective communications where possible, and keep the lessons learned from the BSP model in mind.
- This probably works best after having obtained some experience with BSPlib.
Differences between BSPlib and MPI

- BSPlib: system optimises. MPI: user optimises.
- BSPlib: small. MPI: large.
- BSPlib is easier for the novice. MPI gives experts more power.
- BSPlib: paternalistic library which steers programming efforts in the right direction. MPI allows many different styles of programming.
Summary: where BSP meets MPI

- Use BSPlib when learning to program in parallel.
- Use MPI later in life.
- Use BSPonMPI if you prefer BSPlib but want the portability of MPI.
- MPI-2 provides one-sided communications.
- Our experimental comparisons were unfair to BSPlib. More testing is needed, also using BSPonMPI.
- The third approach may be the best: write in BSPlib, but be prepared to convert to MPI. You may never need to!