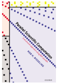
BSP Benchmarking

(PSC §1.5–1.7)

Benchmarking: art, science, magic?

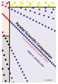
"There are three kinds of lies: lies, damned lies, and statistics" (Benjamin Disraeli, 1804–1881)

- Benchmarking is the activity of comparing performance.
- Computer benchmarking involves running computer programs to see how certain computer systems perform. This checks both the hardware and the system software.
- Often, the benchmark result is obtained by ruthless reduction of a large quantity of data to one statistical figure, the flop rate.



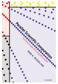
Sequential benchmarking

- Already for sequential computers, benchmarking is difficult, for instance because different programs can run at very different speeds on the same machine.
- Reaching only 10% of the peak rate of a computer is quite common. No one is embarrassed. Hush!
- Highest rates are obtained by algorithms that use matrix–matrix multiplication, such as implemented in the BLAS level 3 operation DGEMM. (BLAS = Basic Linear Algebra Subprograms).
- Lowest rates are obtained for scalar operations, which involve single numbers, not vectors or matrices.
- A reasonable intermediate rate is obtained for vector–vector operations, such as the BLAS level 1 operation DAXPY, defined by $\mathbf{y} := \alpha\mathbf{x} + \mathbf{y}$. We use this operation for sequential benchmarking.

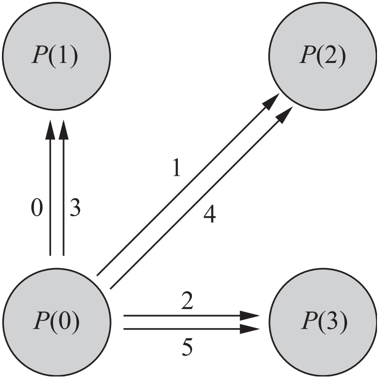


BSP benchmarking

- We must be ruthless, but a single number will not work. Thus we measure: r for computation, g for communication, and l for synchronisation.
- The aim is to obtain useful values of r, g, l that help us in predicting performance of algorithms without actually running an implementation.
- Most of our troubles in this endeavour come from the difficulty of sequential benchmarking.
- A cache is a small memory close to the CPU that stores recently accessed data. There may be a tiny primary cache, a larger secondary cache farther away, etc.
- Computations in primary cache are much faster than others. We may have to distinguish rates r_1, r_2, etc. (but we won't).



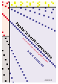
Communication pattern for BSP benchmark program



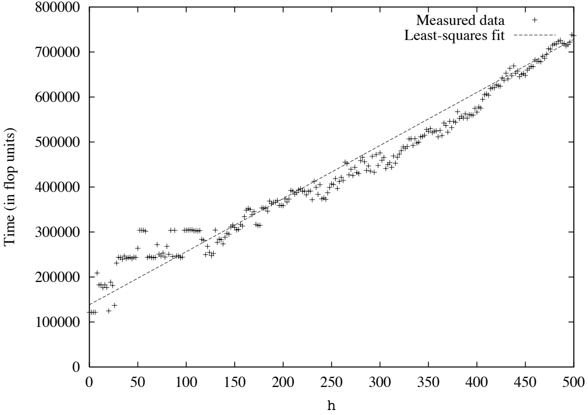
$P(0)$ sends data to $P(1)$, $P(2)$, $P(3)$, $P(1)$, $P(2)$, $P(3)$. The other processors also send data in this cyclic fashion.

Full h-relation

- We measure a full h-relation, where every processor sends and receives exactly h data.
- Our intentions are the worst: we try to measure the slowest possible communication. We put single data words into other processors in a cyclic fashion.
- This reveals whether the system software indeed combines data for the same destination and whether it can handle all-to-all communication efficiently. This is after all the basis of BSP!
- ‘Underpromise and overdeliver’ is the motto: actual communication performance can only be better. We call the resulting g obtained by our benchmarking program `bspbench` pessimistic.
- The Oxford BSP toolset has another benchmarking program, `bspprobe`, which measures optimistic g-values.



Time of an *h*-relation on two connected PCs



Two 400 MHz Pentium II PCs, both running Linux, connected by Fast Ethernet (100 Mbit/s) and a Cisco Catalyst switch.
$r = 122$ Mflop/s, $g = 1180$, and $l = 138324$.

Least-squares fit

- Two measurements would suffice for obtaining a straight line, but we want to use all data available in an interval $[h_0, h_1]$.
- We minimise the error

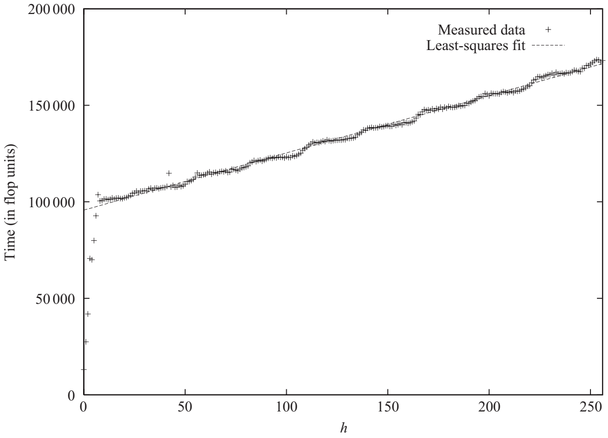
$$E_{\mathrm{LSQ}}(g, l) = \sum_{h=h_0}^{h_1} (T_{\mathrm{comm}}(h) - (hg + l))^2.$$

- The best choice for g and l is obtained by setting

$$\frac{\partial E}{\partial g} = \frac{\partial E}{\partial l} = 0$$

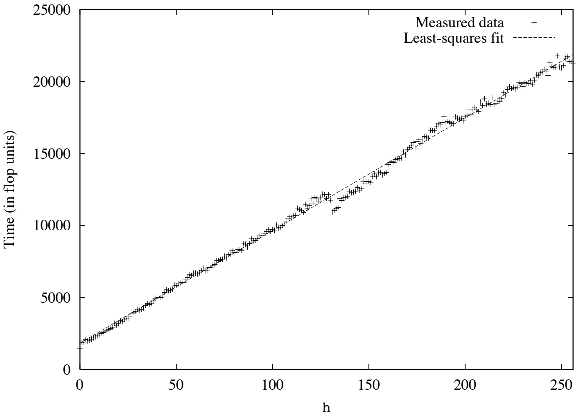
and solving the resulting 2×2 linear system.

Time of an h-relation on an 8-processor SGI Origin

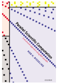


Silicon Graphics Origin 2000, $r = 326$ Mflop/s, $g = 297$, and $l = 95\ 686$. Compiler plays tricks: measured value of r too high. Choose h_0 and h_1 judiciously. Here, $h_0 = p$.

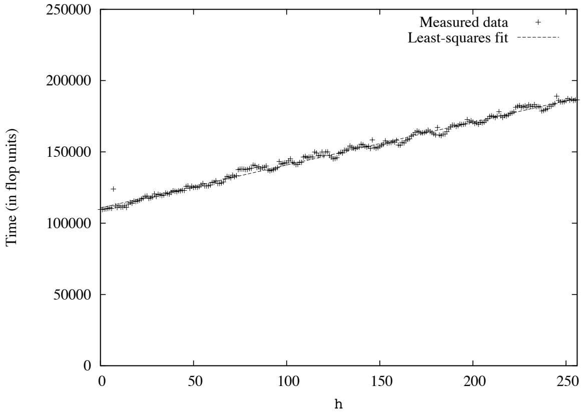
Time of an h-relation on a 64-processor Cray T3E



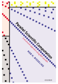
$r = 35$ Mflop/s, $g = 78$, and $l = 1825$.
Sending more data takes less time (cf. $h \approx 130$). Weird!
Explanation: switching to a different data packing mechanism (from short messages to long messages).



Time of an h-relation on an 8-processor Bullx DLC system



$r = 9457$ Mflop/s, $g = 301$, and $l = 110682$.
Supercomputer Cartesius at SURFsara in Amsterdam.
Bullx DLC B710 Blades system.
Number 225 on Top500 (June 2014).



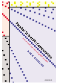
bspbench: initialising the communication pattern

```
for (i=0; i<h; i++){
    src[i]= (double)i;
    if (p==1){
        destproc[i]=0;
        destindex[i]=i;
    } else {
        /* destination processor is one
           of the p-1 others */
        destproc[i]= (s+1 + i%(p-1)) %p;

        /* destination index is in
           my own part of dest */
        destindex[i]= s + (i/(p-1))*p;
    }
}
```

bspbench: measuring the communication time

```
bsp_sync();
time0= bsp_time();

for (iter=0; iter<NITERS; iter++){
    for (i=0; i<h; i++)
        bsp_put(destproc[i], &src[i], dest,
                destindex[i]*SZDBL, SZDBL);
    bsp_sync();
}
time1= bsp_time();
```

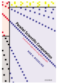
Adjust `NITERS` to obtain an accurate measurement, without waiting forever.



Comparing BSP parameters ($p = 8$)

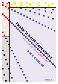
		(flop)		(μs)	
Computer	r (Mflop/s)	g	l	g	l
Cray T3E	35	31	1193	0.88	34
IBM RS/6000 SP	212	187	148212	0.88	698
SGI Origin 2000	326	297	95686	0.91	294
Bullx DLC B710	9457	301	110682	0.03	12

- Machines become obsolete quickly. The first three machines have in the mean time been replaced by faster successors. The Bullx machine is modern (2014).
- Other new machines will be benchmarked in the laboratory class of this course.



Advice from the trenches

- Always plot the benchmark results. This gives insight in your machine and reveals the accuracy of your measurement.
- Be suspicious of artefacts. Negative g values may occur if g is small and l is huge. In that case, the least-squares fit does not give an accurate g and you have to enlarge the measurement interval $[h_0, h_1]$.
- Run the benchmark at least three times. If the best two runs agree, you can be reasonably confident.
- Parallel computers are like the weather: they change all the time. Always run a benchmark program before running an application program, just to see what machine you have today. (Think of: a new compiler, faster communication switches, Challenge Projects that gobble up network resources, and so on.)



Summary

- Benchmarking is difficult.
- Machines have quirks, surprises are plenty, and measurements are often inaccurate.
- With all these caveats, it is still useful to have a table with r, g, l values for many different machines.
- This table should be kept up to date to reflect new architectures appearing. You can do it! (Similar to the LINPACK benchmark used to determine the Supercomputer Top 500.)
- BSP benchmarking can be done using BSPlib (`bspbench`, `bspprobe`), but also MPI-1 (`mpibench`).