Introduction to the BEOWULF Design

Robert G. Brown
Duke University Physics Department

Parallel Computation

- **"Tasks" typically have both serial and parallel components.**
- **Parallel components are subtasks that can be accomplished at the same time with or without active communication. Serial components have to complete one at a time before work can proceed further.**
- **Serial subtask completion time is (usually) "irreducible" (although it can expand!).**
- **Parallel subtask completion time under ideal circumstances scales like $1/N$ where N is the number of parallel tasks undertaken at the same time. "Many hands make light work".**
- **Parallel subtasks typically requires communication between the subtasks. This communication time adds to to the total and can take more or less time than the work itself**
- **All this is made formal in Amdahl's Law and quantitatively corrected in books on parallel computation.**

The "rate" at which a computer does a given piece of "work" is:

$$R = \frac{\mathrm{W}}{T} \tag{1}$$

Putting in the time required for serial and parallelizable work explicitly:

$$R(1) = \frac{\mathrm{W}}{\mathrm{T_s} + \mathrm{T_p}} \tag{2}$$

where the "1" indicates the number of CPUs. Splitting up the parallel work P ways the rate becomes as much as:

$$R(P) = \frac{\mathrm{W}}{\mathrm{T_s} + (\mathrm{T_p}/\mathrm{P})} \tag{3}$$

and the speedup is thus:

$$R(P)/R(1) = \frac{(\mathrm{T_s} + \mathrm{T_p})}{(\mathrm{T_s} + (\mathrm{T_p}/\mathrm{P}))} \tag{4}$$

which is *Amdahl's Law*. This strictly limits the speedup of a parallelized program. It is, however, usually not pessimistic enough.

For example, we have to account for the time spent communicating between processors, the time spent setting things up, and changes in the times to perform various tasks with different algorithms. Defining things like:

T_s The original single-processor serial time.

T_{is} The (average) additional *serial* time spent doing things like IPC's, setup, and so forth, per processor, in all parallelized tasks.

T_p The original single-processor parallizable time.

T_{ip} The (average) additional time spent by each processor doing just the setup and work that it does in parallel. This may well include idle time, which is often important enough to be accounted for separately.

we can obtain improved estimates of the speedup:

$$T_{\rm tot}(P) = T_s + P * T_{is} + T_p/P + T_{ip}. \tag{5}$$

or

$$\frac{R(P)}{R(1)} = \frac{T_s + T_p}{T_s + P * T_{is} + T_p/P + T_{ip}}. \tag{6}$$

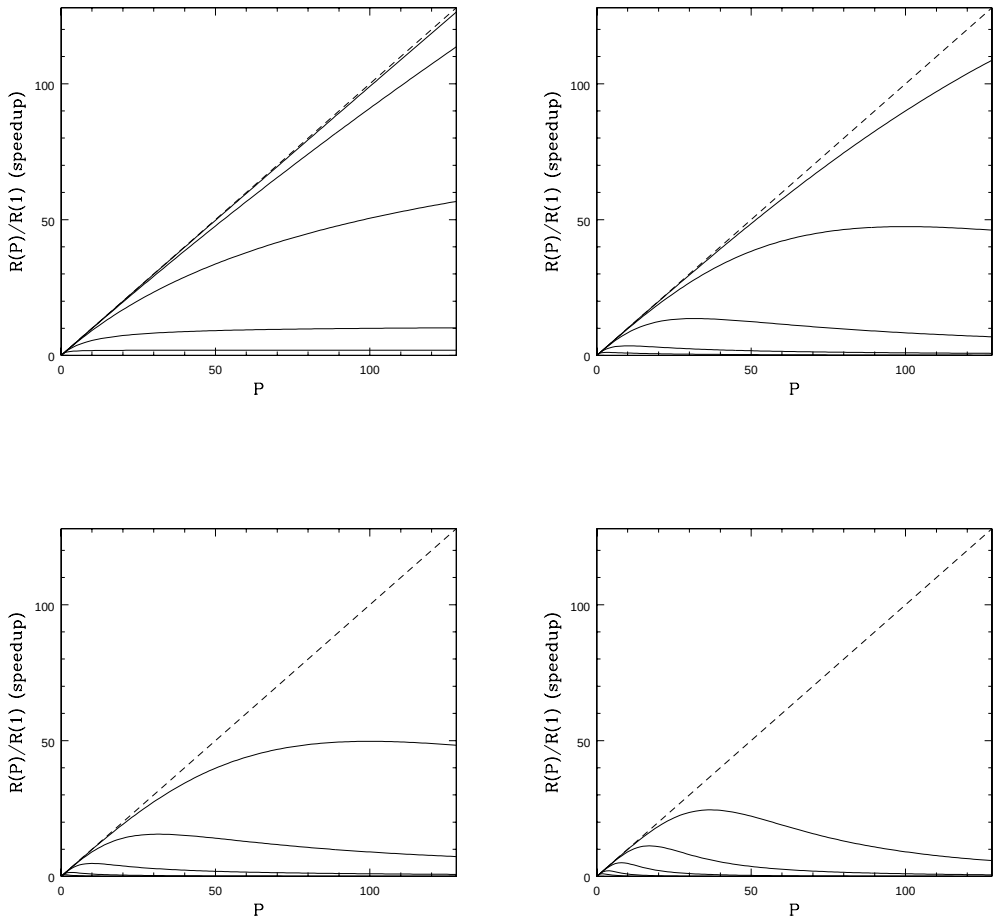
Code Granularity

- **If $t_{\text{computation}} \gg t_{\text{communication}}$, coarse grained. P completely independent jobs are "embarrassingly coarse grained" or "embarrassingly parallel".**
- **If $t_{\text{computation}} > t_{\text{communication}}$ (but not tremendously so) medium grained.**
- **If $t_{\text{computation}} = t_{\text{communication}}$ or less fine grained.**

Granularity typically is controllable. Faster networks, scaling of computation to communication as a function of problem size, CPU speed, program organization all control variables.

Fine grained tasks are "bad" for scaling to many nodes N. Coarse grained tasks are "good".

In all the figures below, $T_s = 10$ (which sets our basic scale, if you like) and $T_p = 10, 100, 1000, 10000, 100000$. In the first three figures we just vary $T_{is} = 0, 1, 10$ for $T_{ip} = 1$ (fixed). Finally, the last figure is $T_{is} = 1$, but this time with a *quadratic* dependence $P^2 * T_{is}$.



Examples

- **Embarrassingly coarse grained: Monte Carlo, data field explorations. Multiple independent jobs with only result collection/summation.**
- **Coarse to medium grained: Monte Carlo, problems on a lattice (where the lattice is partitioned among nodes with short range communications), lattice gauge theory.**
- **Medium to fine grained: Cosmology, molecular dynamics with long range interactions, hydrodynamics, computational fluid dynamics.**

Note that this (oversimplified) view has granularity decreasing as systems spend more and more time communicating data so that the various nodes can take a step. Thus granularity is a function of the computer system as much as it is the code or problem!

Note also that the goal of system design is to pick the cost/benefit optimal parallel computation system that allows YOUR problem to be effectively and efficiently coarse grained.

Beware nonlinearities! CPU/cache/memory/disk bottlenecks can create "superlinear speedup" and violate Amdahl's Law!

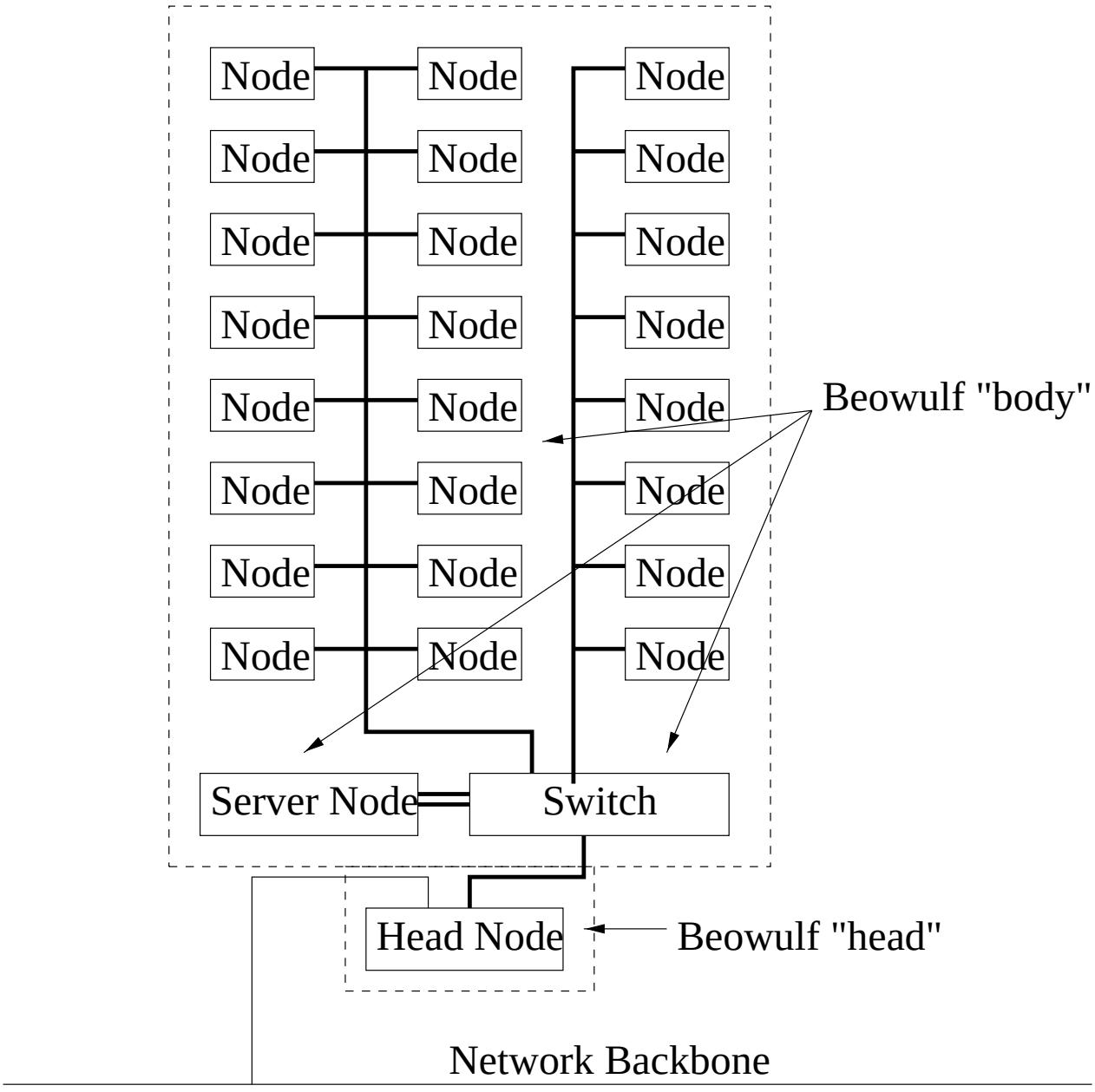
Designs: NOW/COW/Beowulf

- **Goal is optimizing overall performance per dollar. The following are appropriate for increasing fineness of program granularity.**
- **NOW (Network of Workstations) + e.g. Mosix or master/slave PVM or perl script permits double usage of all CPUs – desktop activities eat tiny fraction of capacity, the rest is all calculation(s). Totally maximizes utility of boxes.**
- **COW (Cluster of Workstations) + Mosix/PVM/MPI protects the network a bit and isolates the compute resource (from humans). A bit more computation power, a bit less desktop utility.**
- **Beowulf (dedicated, single headed COW) + Mosix/PVM/MPI + custom software. A totally isolated COW with (usually) a private network and a single head.**

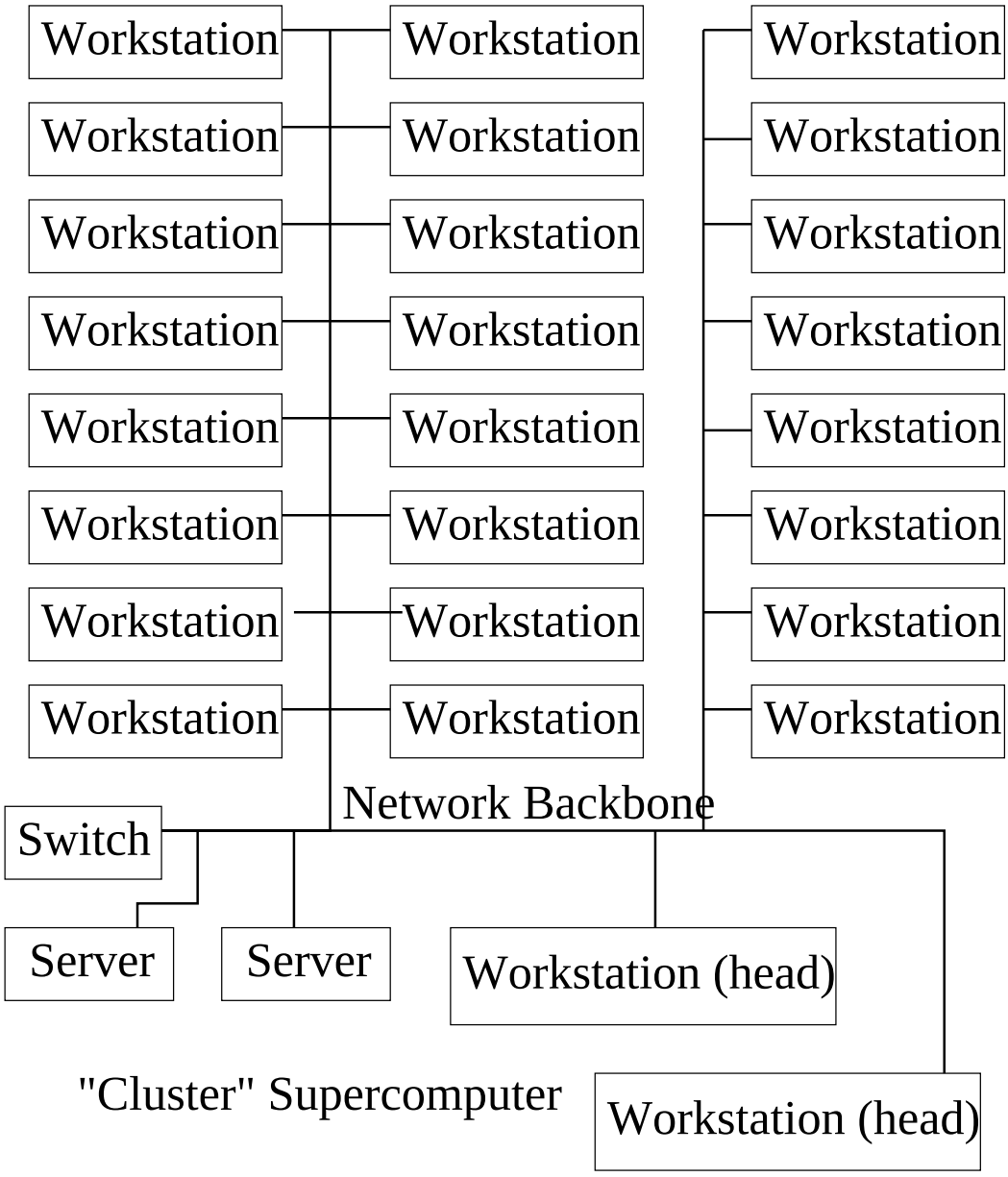
These are suitable for increasingly fine granularity, at increasing cost and decreasing general purpose utility.

Schematics for the general designs follow, first a "true beowulf" and then a workstation cluster.

A True Beowulf



A Workstation Cluster



Intel Beowulf or Cluster Cost/Benefit

- **$500-$2500/node (varying CPU, memory, disk, communications, graphical head).**
- **Absolute maximum (IMHO) raw CPU performance/price is stripped Celeron nodes – 500 MIPS at $500/node. Good for many/most CG problems.**
- **Ranges up to 1 GHz PIII's with big memory, big disk, graphical head, multiple fast ethernets and channel bonding. Myrinet possible, add $1500/node**
- **Best for medium to coarse grained problems. Ideal for embarrasingly parallel problems, e.g. - SETI, RC5/DES, multiple independent (Monte Carlo) simulations.**
- **Tools like Mosix permit "the network to be the computer" as never before.**
- **Strictly COTS components (e.g. – www.intrex.com under "parts"). Cheap and easily maintained.**
- **Do it yourself (easy) or buy turnkey operation (Paralogics, Alta Systems, VA Linux).**

Alpha/Myrinet Beowulf Cost/Benefit

- **$5000-$8000/node (Base 21264 alpha might cost $3500, Myrinet $1500 give or take, plus hardware variations in memory and so forth).**
- **Absolute maximum (IMHO) raw CPU and network performance – nodes 3x faster in floating point, 10x faster in network IPC, lower OS overhead.**
- **Makes most Intel "fine grained" problems coarse grained.**
- **Excellent scaling (to 64 processors and beyond) on fine grained code.**
- ***Beats* a T3E, SGI Origin, or IBM SP2 in raw performance on a per-processor basis for fine grain parallel code (e.g. cosmology, weather apps) at a fraction (perhaps 1/10) of the cost.**
- **Do it yourself or buy turnkey operation (from Greg Lindahl of HPTi, or perhaps Michael Huntingdon of spcnet).**

"Big Iron"

- **Bad: Not infrequently, can build a beowulf of equivalent power for the *maintenance costs alone* of a T3E, SP2, or Origin.**
- **Bad: Is unlikely to be better and is often going to be worse than a suitable beowulf design regardless of the cost.**
- **Bad: Difficult and expensive to maintain and upgrade – depreciation rates of 50%/year or even more are standard fare, given Moore's Law.**
- **Bad: Proprietary, single source vendor, expensive software, high infrastructure costs. *Not* COTS!**
- **Good: Their IPC's can be marginally superior for certain classes of problems. Sometimes.**
- **Good: They keep a whole staff of computer people off the street caring for and feeding a big iron system and yields huge profits to the companies that make them. This keeps our taxes low, except when government grants are used to buy them.**

Conclusions

- **MOST if not ALL parallelizable problems can be run most cost-efficiently on a COTS compute cluster (beowulf).**
- **Worth learning how to design "properly". See references following.**
- **Beowulfish clusters generally at least order of magnitude more price productive, often yield staggering productivity efficiency...**
- **...so literally everybody is buying or building them.**
- **Beware Bottlenecks! Ditto nonlinearities in general, synchronization issues. Your mileage may vary!**

References and Resources

- **http://www.beowulf.org**
- **http://www.phy.duke.edu/brahma/**
- **"How To Build a Beowulf", by Sterling, Becker, et. al.**
- **http://www.phy.duke.edu/∼rgb/beowulf.pdf (private snapshot *my* book on beowulfery).**
- **http://www.phy.duke.edu/∼rgb/beowulf_intro.pdf (this talk).**
- **Online book on designing parallel programs by Ian Foster at Argonne National Labs, http://www-unix.mcs.anl.gov/dbpp/**
- **"Highly Parallel Computing", by Amalsi and Gottlieb.**