CS-E4690 – Programming Parallel Supercomputers

Taxonomies and definitions

Maarit Korpi-Lagg <u>maarit.korpi-lagg@aalto.fi</u>





Flynn's taxonomy



Instruction level parallelism



Low level operational model of a modern CPU

SISD: Scalar vs. superscalar processor

Pipelining: MISD-like setup



Data parallelism



Vectorization

SIMT



Task parallelism



Top level operational model of modern supercomputer application

Adding concurrency

SPMD



Modern hybrid architectures

Combining all Flynn's taxonomies in a way or another

How to build applications for such systems?

[1] A recent review of processor types used in HPC infras for those who are interested.







An array of (sequential) computers



Multicore shared-memory chips







Interconnect

Infiniband protocol and connector technology; Ethernet protocol. Important properties affecting the performance (global bandwidth and latency)

- **Topology** (How are the links in between compute nodes organized; who can connect to who, through whom)
- Connection type (How is the processing unit connected to the network)



Topology

Physical networks + virtual mappings of the processes when designing a parallel program

- **Degree:** number of links from a node
- Diameter: maximum number of links between nodes over a path with minimal distance (worst case routing distance)
- Average distance: number of links to a random node
- **Bisection:** minimum number of links that divide the network into two equal halves (can estimate worst case bandwidth)
- Bisection bandwidth: Bisection x link bandwidth

Minimize diameter, maximize bisection



Topology: examples Bus Control of the section of



Diameter: 3 (n/2) Bisection: 2 Number of switches: n



Topology: examples

2D torus

2D mesh









To improve further, multilevel networks



Topology: examples

Butterfly



Diameter: 3 (ln(n)-1) Bisection: 8 (n/2) Number of switches: 32 (n*(ln(n)+1))

Cost: the more links and switches (large hop count), the more resources it takes to build and operate

Topology: modern HPC

Fat tree





Topology: modern HPC

Dragonfly



Fully connected graph

Minimizes diameter and maximises bisection



Topology: modern HPC



What would then this be?

Mahti@CSC, image credit CSC



Interconnect

Summary of current HPC interconnects

- Topology (Dragonfly, fat tree, torus, all sorts of combinations at multiple layers)
- Connection type (Currently most networks are multi-level, switches can handle an order of hundred of ports)
- Latency (1-2microsecs)

Ito University

Bandwidth (Nowadays around 100Gbit/s-200Gbit/s achieved through multiple lanes)

[1] A recent review of interconnect status for those who are interested.

20

Interconnect

How much (as a user) do you care about the interconnect topology?

- Thanks to libraries such as MPI, not much. Why? Should one?
- Jobs are usually small in comparison to the scale of the system (fit into a chunk or island).
- On this local scale, the job schedulers do a good job.
- Larger simulations challenge all this, and we are heading towards exascale computing.
- Hence, learning about topologies is not in vain!



Bhatelé & Kalé (2009) IBM BG/P using messages between equidistant pairs

Latency (us)



Latency vs. Message Size: With varying hops (8 x 8 x 16)

Performance models without interconnect: RAM and PRAM

- If there was only shared memory...
- $T_{RAM} = N_C + N_M$
- N_c the number of instructions completed
- N_M the number of loads/stores from/to the memory
- Communication to other distant nodes is not an issue
- For sequential algorithms, still valid, but requires extensions to take into account multi-level caches.
- PRAM an extension to multiple processors



Latency-bandwidth performance model ($\alpha\beta$, used since the 1970's)

 $T_{LB} = \alpha + n(\beta + \gamma);$

 α = Latency (start up cost of communication)

- β = Time cost per unit message length sent (bandwidth cost)
- γ = Time consumed in actual computation
- *n* = Message length
- Both receiver and sender block
- No multiple messages allowed
- Does not allow for overlap (concurrency) in communication and computation.



BSP model (Valiant et al. 1990)

 Bulk synchronous applications that perform supersteps

•
$$T_{BSP} = \max_{i=1}^{p}(\omega_i) + \max_{i=1}^{p}(gnh_i) + l$$

p number of processors

h number of messages, n their length g bandwidth throughput

l barrier cost

- Topology is not accounted for
- Has many applications, including MapReduce.

Interested knowing more: Pace (2012)



LogP model (Culler et al. 1993)

Asynchronous messages g ۲ $T_{logP} = 2o + L + (n-1)g$ P0 send send *n* is the data size o is the overhead (processor) L is the network latency receive P1 · $g \ge o$ is the gap between two sends or receives

What is required in practise?

Not-wanted situation: communication is not totally overlapped with computations.

Superstep



Asynchronous communication-compute performance model for real applications (ACC) [3]

In reality, the "gap" does not only come from "preparations" to communicate, but from actual computations taking quite some time. These can be overlapped with comms.

Let us denote **latency of computations as** τ_W , when performing W operations on data items, and that of **communications as** τ_0 , when communicating Q data items. π is the operational capability of the hardware as data item updates per second, and β the rate at which data items can be communicated. The latencies of computation and communication are therefore, $\tau_W = W/\pi$ and $\tau_O = Q/\beta$ respectively. Let us further assume that there is a portion of computations that cannot be overlapped (made concurrent) with the communication; let this fraction be τ_0 The total latency is determined by the non-concurrent parts of the code

 $\tau = \max(\tau_W, \tau_Q) + \tau_0$



ACC model cntd

Trivial limit: τ_0 dominates (rather a rare condition).

When τ_0 is small, there are two interesting limits:

1) When there is a lot to compute, and little to communicate, $\tau_W > \tau_Q$, we are in the compute-bound limit.

2) When there is a lot to communicate, and little to compute, $\tau_W < \tau_Q$, we are in the communication-bound limit (where all the other performance issues of memory exchange/interconnect kick in).

Goal: always to stay in limit 1), never run an application in limit 2).



Amdahl's law

One way of determining speedup from parallelism:

Let f be the fraction of computations performed sequentially. The fraction that can be done in parallel is (1 - f). The time that it takes, for a fixed problem size, to execute all the computations with P processing units is then

$$T_P = f + \frac{(1-f)}{P}$$

The speedup is $\frac{T_1}{T_P} = \frac{1}{f + (1-f)/P} < \frac{1}{f}$; limited by the sequential part



Amdahl's law

Aalto University



Strong scaling

- The problem size is kept fixed, the number of PUs is increased, and the execution time is monitored.
- Due to the serial part of the code, the ideal linear scaling (when you double PUs, the computing time halves) saturates to a level $\frac{1}{f}$, after which increasing PUs does not make sense.
- Follows from Amdahl's law.
- Strong scaling to large number of PUs is very challenging to retain.



Gustafson's law

Another viewpoint on speedup from parallelism:

If the workload is growing, the serial fraction of the program does not limit speedup if the parallel part scales well. The speedup can be written as the ratio of the workloads

$$\frac{W_P}{W_1} = f + (1-f)P.$$



Weak scaling

- Follows from Gustafson's law.
- The problem size is not kept constant, but increased so that each PU has a constant workload, and one monitors the execution time.
- Perfect scaling up to many more PUs is easier to maintain.
- Usually both strong and weak scalings are required to be shown when testing the parallel performance of a code.



Stencils

- Recurring update patterns (iterative stencil loops, ISL) of arrays
- Nearly everywhere, can be of any shape and complication



2D von Neumann



2D Moore





Stencils

- **Order** of stencil: how many neighbors in a certain cardinal • direction it needs for updating the central point. (Not useful for asymmetric stencils)
- **N-point stencil:** N is the total number of points (including itself) ٠ it needs for updating the central point PU1 PU3

2nd order 9

Challenge parallelism: •

> Tasks are not independent, as they need data from other PUs to complete the update



ISL: Halo exchange

The points that need to be exchanged form the so called halo. The size of the halo depends on the order and structure of the stencil.

Example: 2nd order, 2D von Neumann stencil, 5x5 computational subdomain Red cells: **edge** points that can be updated when the halo points are received





Green cells: Interior points that can be updated without data exchange



ACC model [3]: example curve for high-order ISLs.





ACC model [3]: example curve for high-order ISLs.





Core reading

[1] Tekin et al., "State-of-the-Art and Trends for Computing and Interconnect Network Solutions for HPC and AI", <u>Prace publications.</u>

[2] Zhang, Y., Chen, G., Sun, G., and Miao, Q. (2007). Models of parallel computation: A survey and classification. Frontiers of Computer Science in China, 1:156–165.

[3] Pekkilä, J. et al. "Scalable communication for high-order stencil computations using CUDA-aware MPI", Parallel Computing. 111, 102904. https://doi.org/10.1016/j.parco.2022.102904



Extra reading

- Valiant, L. G. (1990). A bridging model for parallel computation. *Communications* of the ACM, 33(8):103–111.
- Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subra-monian, R., and von Eicken, T. (1993). LogP: towards a realistic model of parallel computation. Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming, 28(7):1–12.
- Bhatelé and Kalé (2009), Quantifying network contention on large parallel machines, Parallel Processing Letters, 19(4), <u>https://doi.org/10.1142/S0129626409000419</u>
- Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.
- Pace, M. F., (2012), "BSP vs. MapReduce", Procedia Computer Science, 9, 246-255, doi:10.1016/j.procs.2012.04.026
 Aalto University School of Science