Making Applications Run Faster By Slowing Down Processes?

Ayesha Afzal, Georg Hager, Gerhard Wellein

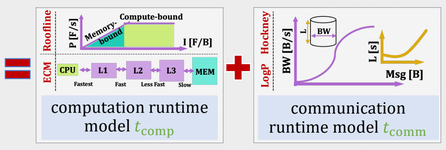
Erlangen National High Performance Computing Center

NHR FAU

Setting the stage: runtime predictions and asynchronicity

Hypothesis

distributed runtime prediction t_{pred} =



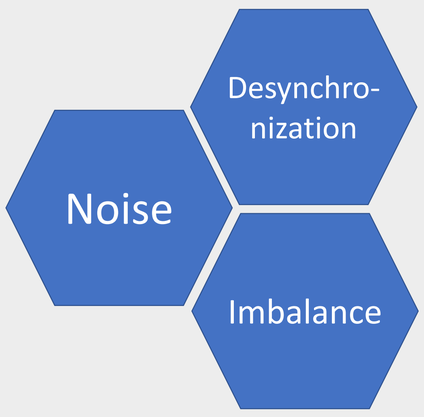
$$t_{pred} = t_{\text{comp}} + t_{comm}$$

Reality

$$t_{actual} = F(t_{\text{comp}}, t_{comm})$$

Encodes complex interplay of noise, system bottlenecks, and overlapping contributions

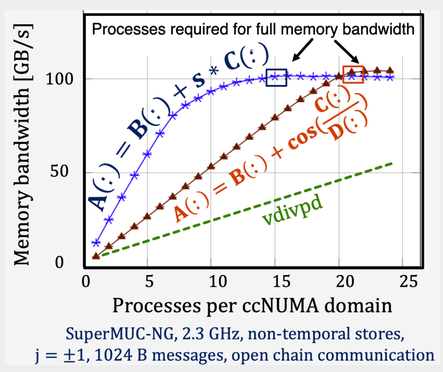
Reasons?



$t_{actual} \not\approx t_{pred}$

Loss of lock-step behavior

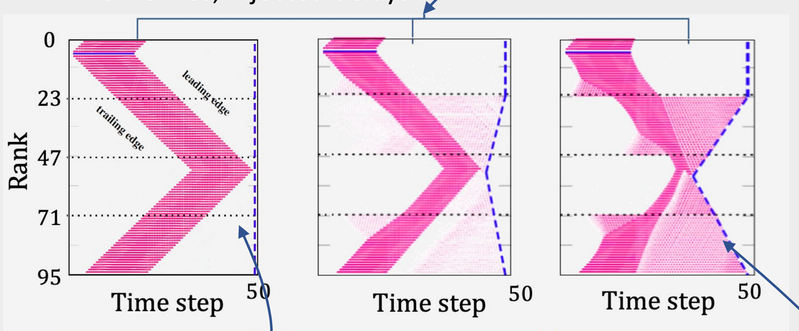
How desynchronization and asynchronicity emerge from bottlenecks and noise



SuperMUC-NG, 2.3 GHz, non-temporal stores, j = ±1, 1024 B messages, open chain communication

Different scaling characteristics on memory bandwidth bottleneck

MPI timelines, injected delays



Different behavior of propagating disturbances (noise)

Back-to-back vdivpd's	A(:)=B(:)+cos(C(:)/D(:))	A(:)=B(:)+s*C(:)
Perfectly scalable	Just saturating	Strongly saturating

Scalable code → irregularities die out → processes go back to sync

Strong saturation → processes leave lockstep → stable desync state → potential communication overlap

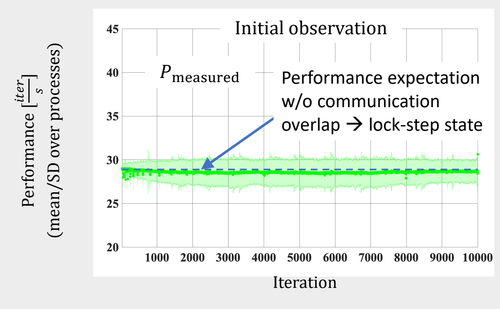
DOI: 10.1007/978-3-030-50743-5_20 DOI: 10.1109/CLUSTER.2019.8890995

Provoking asynchronicity by repeated injection of noise

Working set 48 GB (2 G array elements) evenly distributed among 360 MPI processes (5 nodes Intel Ice Lake 36c, NHR@FAU "Fritz" system), communication volume 1 MB to and from $P_{i\pm1}$ in closed-chain topology,

```
for(iter=1; ;++iter) {
  for (int i=0; i<arrayElements; i++)
     A[i] = B[i] + s * C[i];
  for (int j = 0; j < 2; j++ ) {
     MPI_Isend(..., &req[j*2]);
     MPI_Irecv(..., &req[1+j*2]);
  }
  MPI_Waitall( 4, &req[0], ... );
}
```

Toy code: MPI-augmented STREAM triad with next-neighbor communication

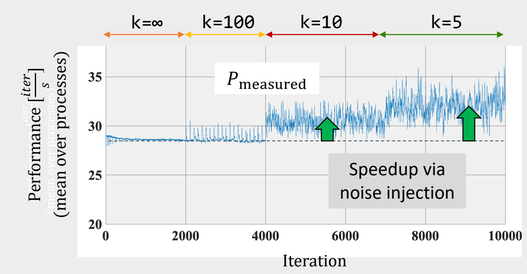


No global synchronization!

How to kick the system out of lock-step?

```
for(iter=1; ;++iter) {
  for (int i=0; I < 2x10^9; i++)
     A[i] = B[i] + s * C[i];
  if(!(iter % k))
     for (int j = 0; j < 10^8; j++ )
        sum += 4.0 / ( 1.0 + j * j);
  for (int j = 0; j < 2; j++ ) {
     MPI_Isend(..., &req[j*2]);
     MPI_Irecv(..., &req[1+j*2]);
  }
  MPI_Waitall( 4, &req[0], ... );
}
```

Extra compute-bound workload every k iterations ("noise")



Provoking asynchronicity by noise injection can improve performance!

The role of synchronizing MPI collectives

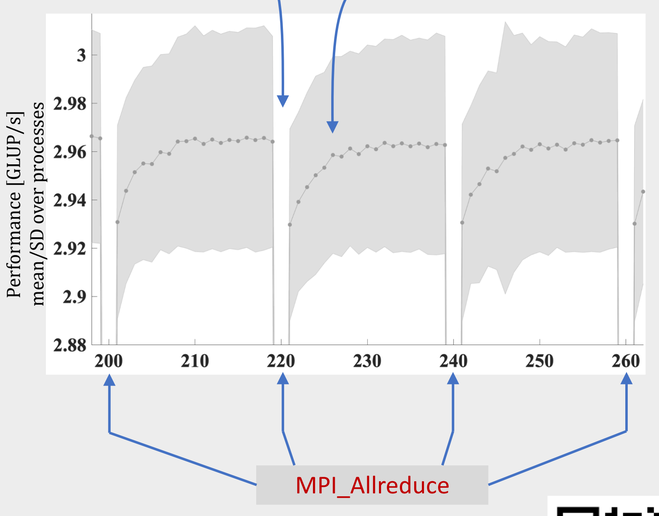
Synchronizing collectives (e.g., MPI_Allreduce) force the program back into lock-step

D3Q19 Lattice Boltzmann, 1D domain decomposition, MPI_Allreduce every 20th iteration , 10 sockets Intel Ice Lake 36c

```
while iter ≤ nIters do
    stream_collide_update (lattice, u_lid, omega);
    set_boundary_condition (u_lid);
    MPI_Isend ; *
    MPI_Irecv ; *
    MPI_Wait ;
    ghost_cells_update ();
    if ((iter % collective_step) == 0) then
      MPI_Allreduce ;
    end if
    swap (local_src_lattice, local_dst_lattice) ;
end while
```

Full sync after collective

Gradual desync and communication overlap



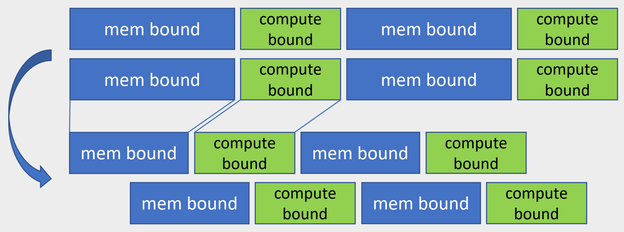
No benefit from noise injections if frequent synchronizations are present!

Caveat: Not all collectives are synchronizing, and implementation does matter

DOI: 10.1007/978-3-030-78713-4_19
DOI: 10.1109/TPDS.2022.3221085

Open questions

Communication overlap is only one option – what about overlapping memory-bound and compute-bound workloads?



Comprehensive theory of parallel program dynamics still lacking – searching for collaborators!

Achievable performance limit?

Instability mechanism for bottlenecked workloads?