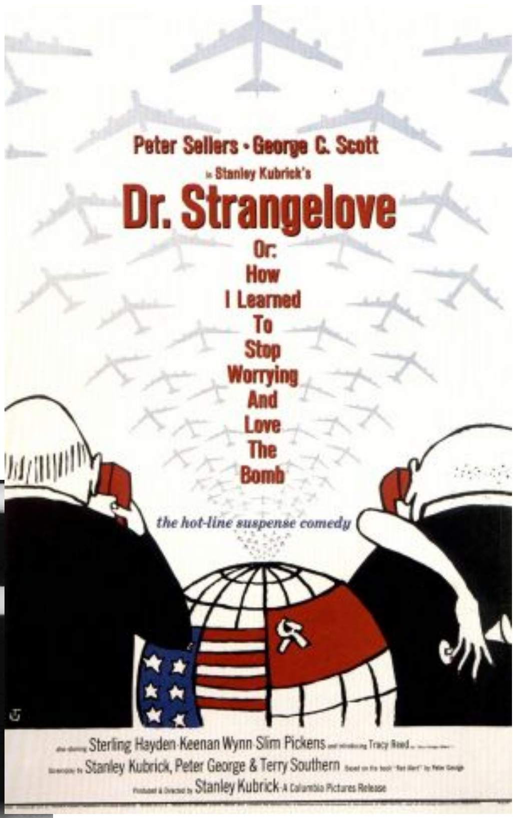
How I Learned to Stop Worrying about Exascale and Love MPI

(Yes, MPI is indeed da bomb!)

Pavan Balaji

Computer Scientist and Group Lead

Argonne National Laboratory


Peter Sellers · George C. Scott
in Stanley Kubrick's
Dr. Strangelove
Or: How I Learned To Stop Worrying And Love The Bomb
the hot-line suspense comedy
Sterling Hayden · Keenan Wynn · Slim Pickens
Stanley Kubrick, Peter George & Terry Southern
Stanley Kubrick · A Columbia Pictures Release

Separating the Myths from Real Concerns

- The race to Exascale started in earnest around 2006/2007
  - E to "outcompete")
  - Te
- Challenges:
  - Business as us
  - Hardware architecture ne
  - Software needs to be built from the gr
    - MPI, OpenMP and other "legacy" softv

"MPI is bulk synchronous"

"MPI cannot deal with manycore systems"

"MPI cannot deal with a

"MPI is not fault tolerant"

"MPI is too static"

*See my previous talk on "**Debunking the Myths in MPI Programming**" for more technical details on these myths*

Current Complaints with MPI

- System architecture too complex and disruptive
  - MPI is too "old school" and assumes a certain architecture
  - MPI cannot run on upcoming architectures
- Some applications becoming irregular/data-dependent
  - No structured pattern, dominated by small messages, asynchronous communication important
  - MPI cannot provide these capabilities

- These claims are not entirely true, but need some thought before dismissing

U.S. DOE Potential System Architecture Targets

System attributes	2012	2017-2018		2023-2024	
System peak	20 Peta	200 Petaflop/sec		1 Exaflop/sec	
Power	9 MW	15 MW		20-30 MW	
System memory	0.7 PB	5 PB		32-64 PB	
Node performance	1.5 TF	3 TF	30 TF	10 TF	100 TF
Node memory BW	25 GB/s	0.1TB/sec	1 TB/sec	0.4TB/sec	4 TB/sec
Node concurrency	O(100)	O(100)	O(1,000)	O(1,000)	O(10,000)
System size (nodes)	20,000	50,000	5,000	100,000	10,000
Total Node Interconnect BW	10 GB/s	20 GB/sec		200GB/sec	
MTTI	days	O(1day)		O(1 day)	

	Current production (e.g., Titan)	*Planned Upgrades (e.g., CORAL)*	*Exascale Goals*

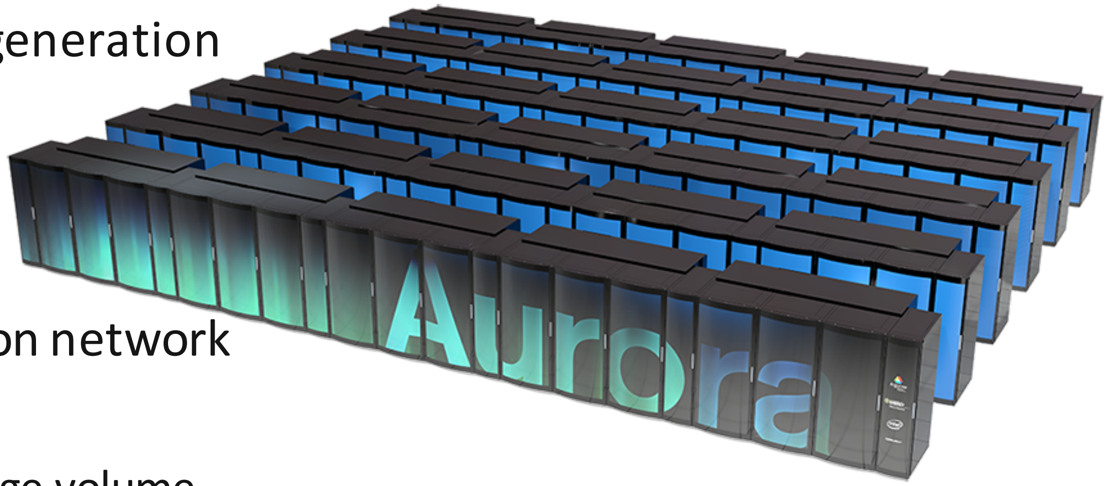
[Based on, but significantly modified from, the DOE Exascale report]

Upcoming US DOE Machines

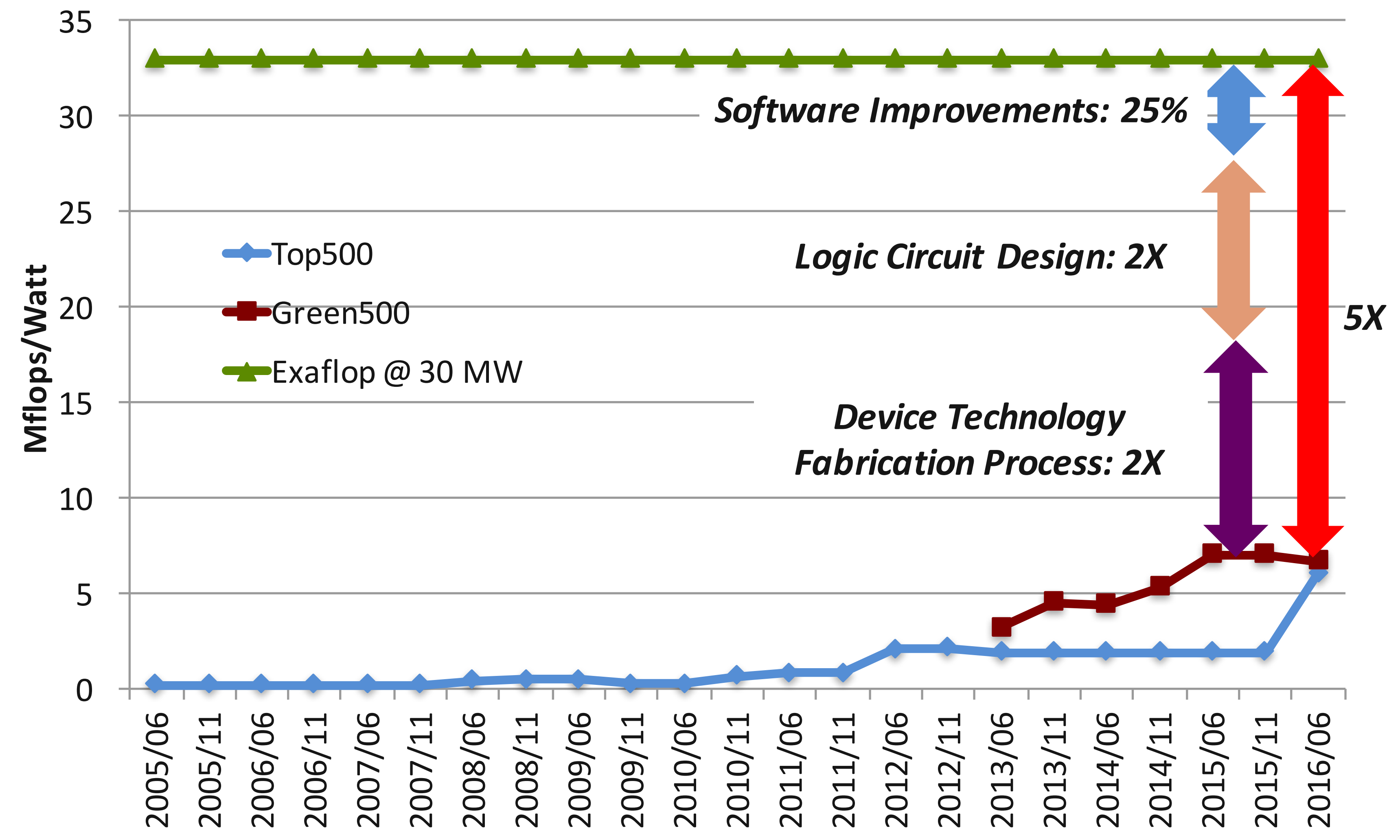
- U.S. is investing in multiple different machines leading up to Exascale machines
  - NERSC-8/Trinity Machines (LBNL, Sandia, LANL collaboration)
    - Cori (2016): NERSC, California (~30 PF)
    - Trinity (2016): Sandia/Los Alamos, New Mexico (~30PF)
  - CORAL machines (ORNL, LLNL, ANL collaboration)
    - Sierra (2017): Livermore, California (150PF)
    - Summit (2017-2018): Oak Ridge, Tennessee (200PF)
    - Aurora (2018-2019): Argonne, Illinois (180PF)
  - APEX (2020): ~300PF
  - CORAL-2 (2023): 1EF

Argonne's CORAL Machine: Aurora

- To be deployed in 2018-2019
- One of the largest systems in the world (100-200PF)
- Based on Intel Xeon Phi (next generation after KNL)
  - Lots of lightweight cores
  - No "host Xeon processor"
- Based on Intel's next generation network fabric
  - Heavily optimized for both large volume data as well as small messages
- Intel is the primary contractor; system integration and deployment by Cray
- ***Applications to primarily rely on MPI or MPI+OpenMP***



On the path to Exascale (assuming Exascale in 2023)

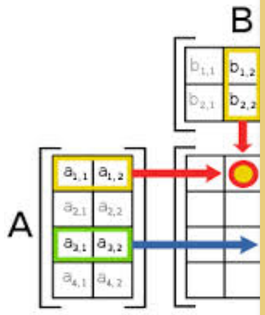


Data courtesy Bill Dally

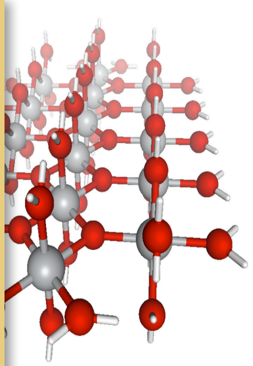
Irregular Computations

- **“Traditional” computations**
  - Organized around dense vectors or matrices
  - Regular data movement pattern, use MPI SEND/RECV or collectives
  - More local computation, less data movement
  - Example: stencil computation, matrix multiplication, FFT

- **Irregular computations**
  - Organized around graphs, sparse vectors, more “data driven” in nature
  - Data movement pattern is irregular and data-dependent
  - Growth rate of data movement is much faster than computation
  - Example: social network analysis, bioinformatics

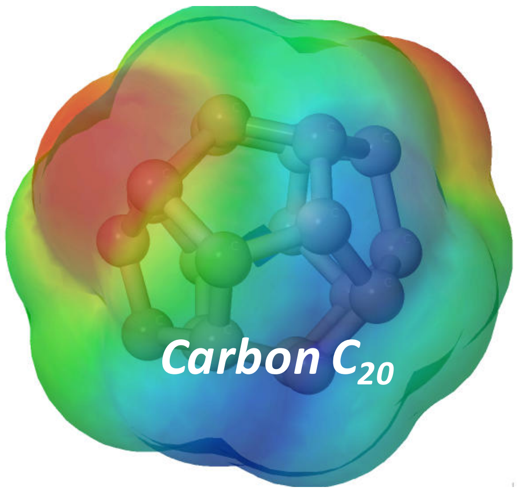


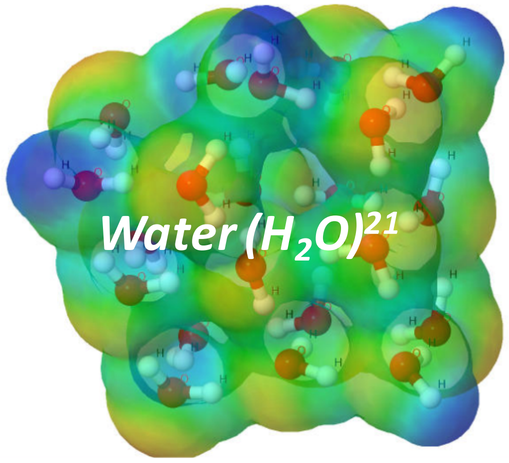
- **New irregular computations**
  - Increasing trend of applications moving from regular to irregular computation models
    - Computation complexity, data movement restrictions, etc.
  - Example: sparse matrix multiplication



NWChem [1]

- High performance computational chemistry application suite
- Quantum level simulation of molecular systems
  - Very expensive in computation and data movement, so is used for small systems
  - Larger systems use molecular level simulations
- Composed of many simulation capabilities
  - Molecular Electronic Structure
  - Quantum Mechanics/Molecular Mechanics
  - Pseudo potential Plane-Wave Electronic Structure
  - Molecular Dynamics
- Very large code base
  - 4M LOC; Total investment of ~200M $ to date

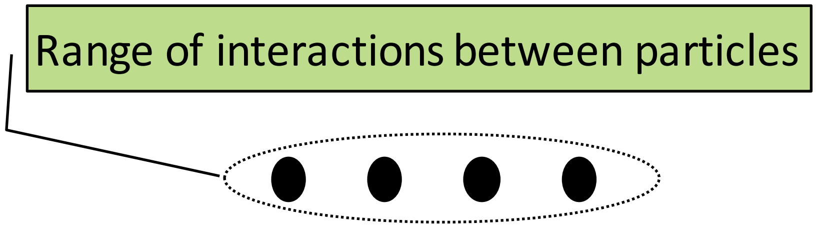


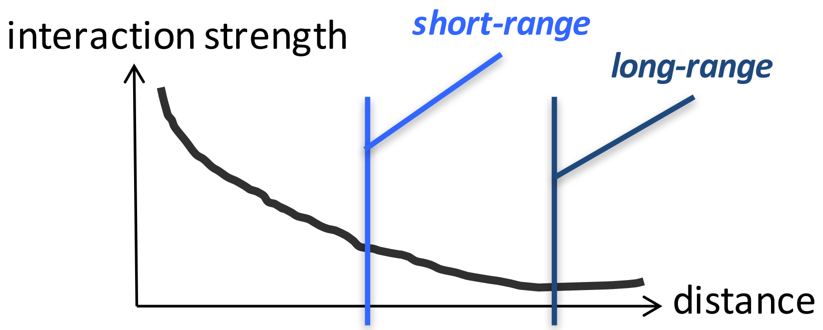


[1] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, W.A. de Jong, "NWChem: a comprehensive and scalable open-source solution for large scale molecular simulations" Comput. Phys. Commun. 181, 1477 (2010)

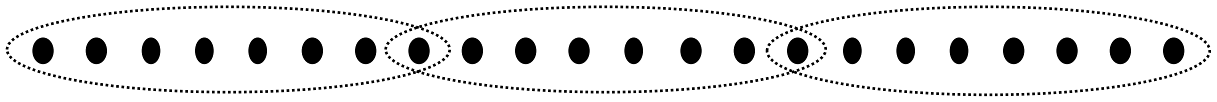
Traditional Coulomb Interactions are Near-Sighted

- Traditional quantum chemistry studies (small-to-medium molecules) lie within the near-sighted range where interactions are dense





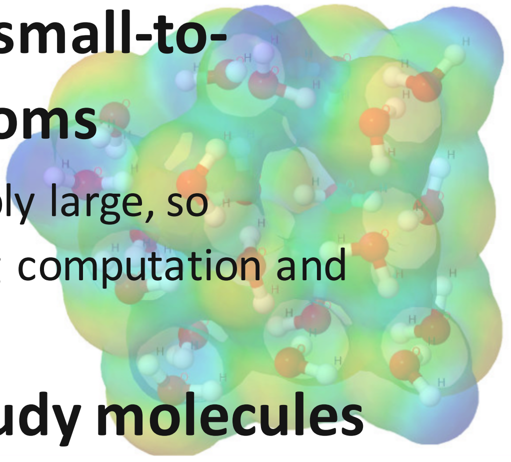
- Future quantum chemistry studies (larger molecules) expose both short-range and long-range interactions



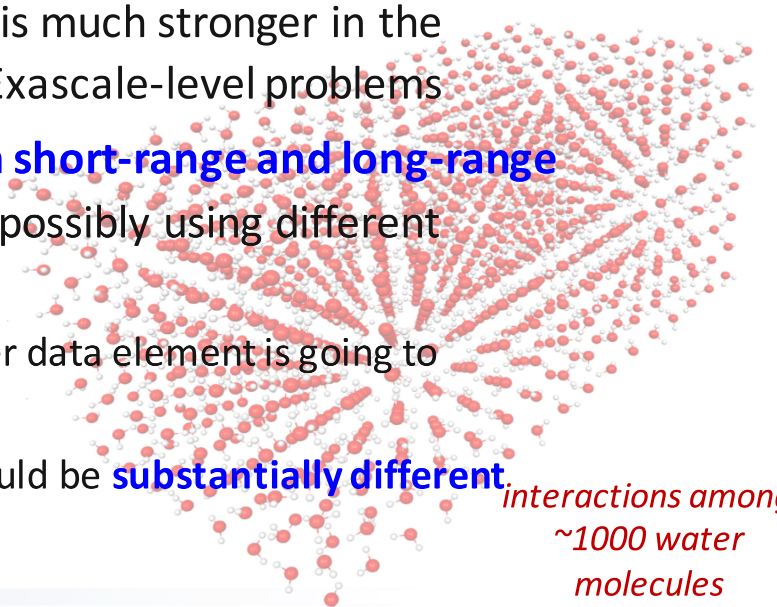
(Note that the figures are phenomenological. Quantum chemistry methods treat correlation using a variety of approaches and have different short/long-range cutoffs.)

Courtesy Jeff Hammond (Intel Corp.)

N-Body Coulomb Interactions

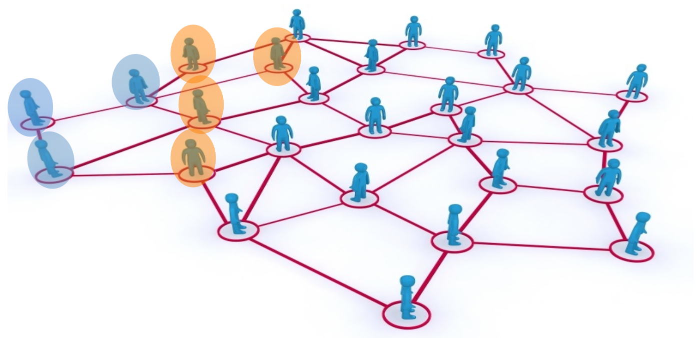

interactions among ~20 water molecules

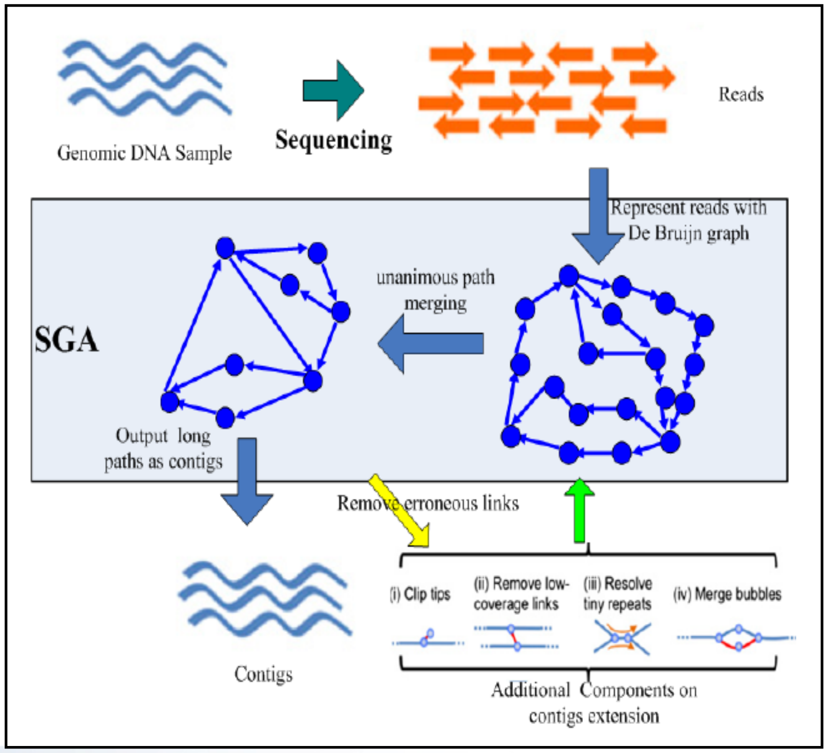
- **Current applications have been looking at small-to-medium molecules consisting of 20-100 atoms**
  - Amount of computation per data element is reasonably large, so scientists have been reasonably successful decoupling computation and data movement
- **For Exascale systems, scientists want to study molecules of the order of a 1000 atoms or larger**
  - Coulomb interactions between the atoms is much stronger in the problems today than what we expect for Exascale-level problems
  - Larger problems will need to support **both short-range and long-range** components of the coulomb interactions (possibly using different solvers)
    - **Diversity** in the amount of computation per data element is going to increase substantially
    - **Regularity** of data and/or computation would be **substantially different**

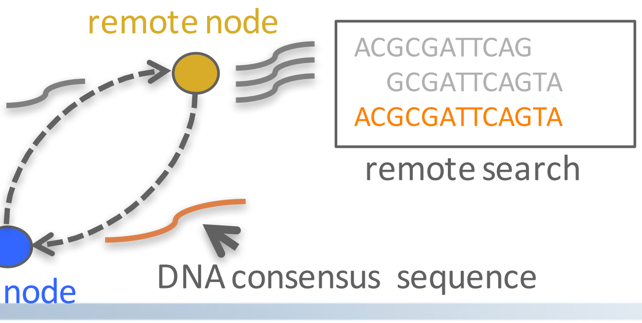

interactions among ~1000 water molecules

Genome Assembly

– **Graph algorithms**
  - Commonly used in social network analysis, like finding friends connections and recommendations
– **DNA sequence assembly**
  - Graph is different for various queries
  - Graph is dynamically changed throughout the execution
  - Fundamental operation: search for overlapping of sequences (send query sequence to target node; search through entire database on that node; return result sequence)

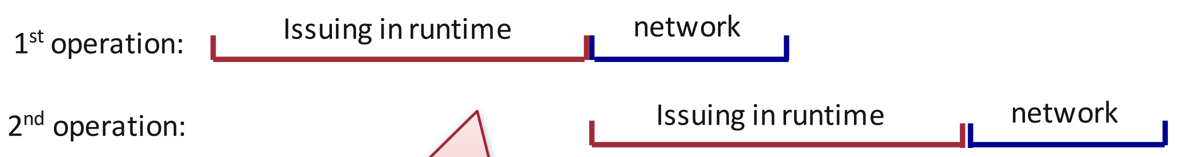


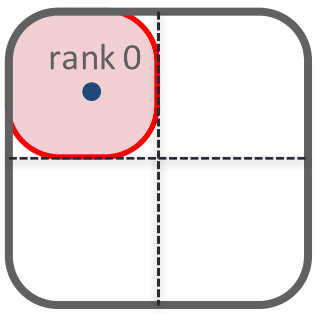

Genomic DNA Sample
Sequencing
Reads
Represent reads with De Bruijn graph
unanimous path merging
SGA
Output long paths as contigs
Remove erroneous links
(i) Clip tips
(ii) Remove low-coverage links
(iii) Resolve tiny repeats
(iv) Merge bubbles
Contigs
Additional Components on contigs extension



Performance Requirement for Network

1~2 cores issue messages to network, network is not saturated

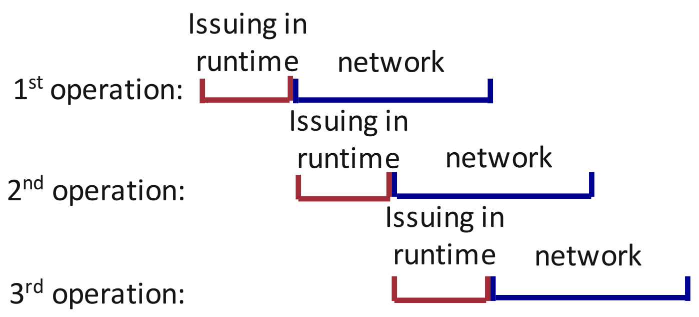


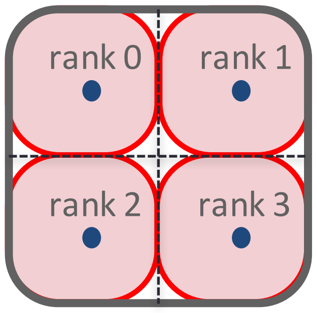


Runtime overhead is the bottleneck!

Optimizing runtime requires new feature from hardware

Increasing #cores that inject messages to network





Large #cores issue messages to network, network can be saturated

Network message rate is the bottleneck!

Single-core performance matters!

MPI Implementation Improvements

High-Level Netmod API

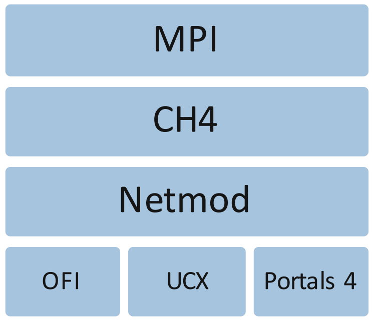
- Give more control to the network
    - `netmod_isend`
    - `netmod_irecv`
    - `netmod_put`
    - `netmod_get`
- Fallback to Active Message based communication when necessary
    - Operations not supported by the network

Provide default shared memory implementation in CH4

- Disable when desirable
    - Eliminate branch in the critical path
    - Enable better tuned shared memory implementations
    - Collective offload

"Netmod Direct"

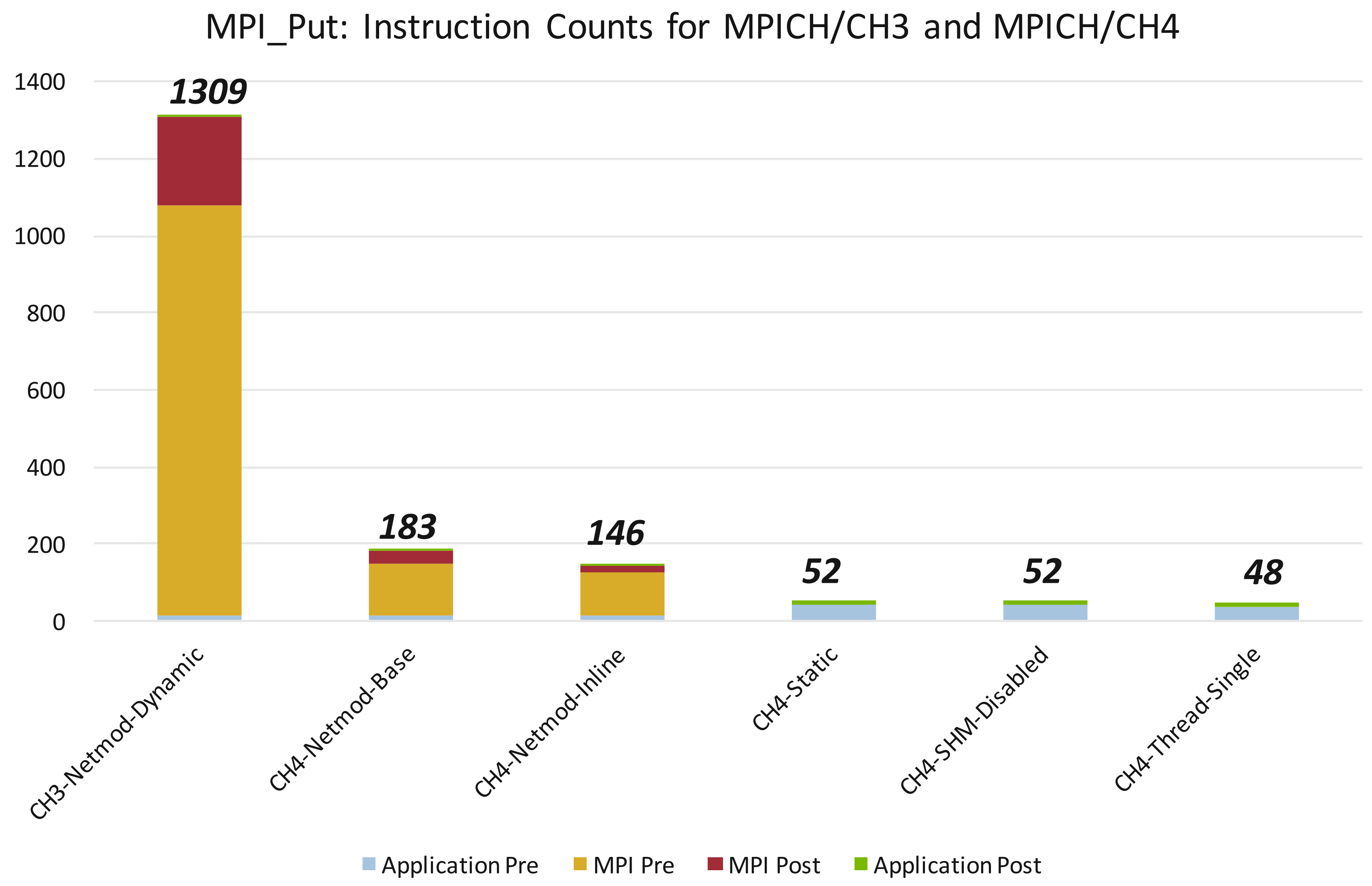
- Support two modes
    - Multiple netmods
        - Retains function pointer for flexibility
    - Single netmod with inlining into device layer
        - No function pointer



No Device Virtual Connections

- Global address table
    - Contains all process addresses
    - Index into global table by translating `(rank+comm)`
- VCs can still be defined at the lower layers

Instruction Counts for CH3 and CH4



Instruction Count Analysis

- Where are my instructions going?
- MPI is a general-purpose runtime layer
  - Cannot quite decide whether its customers are application developers or library writers
- E.g., MPI_PUT is a single function call for many cases

```
MPI_Put(void *origin_addr, int origin_count,
        MPI_Datatype origin_dtype, int target_rank,
        MPI_Aint target_disp, int target_count,
        MPI_Datatype target_dtype, MPI_Win win)
```

MPI_PROC_NULL

- A branch to check for the PROC_NULL case cannot be avoided
  - Additional branch to check for this
- General model to fix such things is through info arguments
  - Does not help in this case
  - Bad idea: info checks can take more time than a regular branch to see if the target rank is PROC_NULL
- Other programming models that do not have the concept of PROC_NULL do not need this branch

```
int MPI_Put(..., target_rank, ...)
{
    if (target_rank != MPI_PROC_NULL) {
      /* do real work */
   }

   return MPI_SUCCESS;
}
```

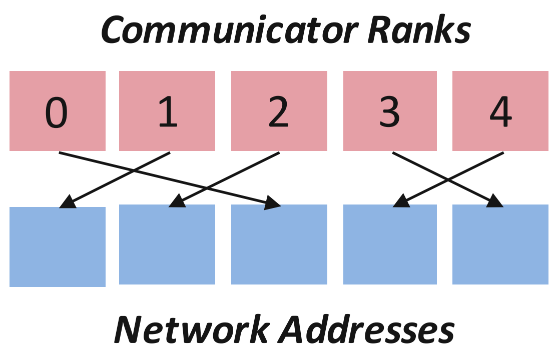
MPI Datatypes

- MPI_PUT is a generic function for any datatype
- MPI implementation needs at least a switch statement to get to what datatype is being transmitted
  - E.g., One integer has the same API as seven derived datatype elements of 3D subarrays
- At least one additional branch needed, likely more
- In contrast, shmem_int_put does not have such a check

```
int MPI_Put(..., origin_datatype, origin_count, target_rank, target_datatype, ...)
{
   if (target_rank != MPI_PROC_NULL) {
     switch(origin_datatype) {
        case MPI_INT:
            if (origin_count == 1)
               network_put_int(...)
            else if (target_datatype is contiguous)  /* bit mask or more */
               network_put_int(...)
            else
               ...
} } }
```

Windows covering arbitrary sets of processes

- Mismatch between application view and network view
  - Communicator is a virtualization of physical processor IDs
  - Target rank in an arbitrary communicator does not make sense to a network; needs to be translated to a global process ID
- Translation has two problems:
  - I need access to internal MPI data structures to find the communicator object
    - At least one pointer dereference; typically two in most implementations
  - I need translate target rank to global ID
    - An O(P) array in most cases, causes another cache miss
    - Can be optimized for the “simple cases”



Offset-based vs. Virtual Address Operations

- MPI_PUT in most cases (except for dynamic windows) requires the user to provide an offset
- MPI implementation then might need to translate this offset to an absolute address if the network does not support it
- For applications that know the target address (e.g., SPMD applications that end up with symmetric allocations), this is an unnecessary check inside MPI
- Offset to absolute address again requires translation:
  - Same problems as the rank lookup
  - Symmetric allocation with WIN_ALLOCATE does not solve the problem
    - I still need to lookup the base address even if it is the same

Recap

- Recommendations:
  - PROC_NULL is an annoyance
    - Added for convenience, but often not worth the effort
    - Applications can easily check for it. No reason for the MPI implementation to check it even if the application never uses it.
  - Datatype-specific operations might be OK to have
    - Function name explosion is not a big deal if the target is library writers, not end users
  - COMM_WORLD (or dup) windows are special
    - This can be mostly handled in the implementation by setting a special bit in the window handle for such windows, but still needs a branch
  - Offset vs. absolute address access needs new function calls
    - MPI_PUT_ABS (we already do this for dynamic windows)
- Good News: MPI-5 will fix all your problems!
  - Evolving standard that incorporates improvements

Take Away

- MPI has a lot to offer for Exascale systems
  - MPI-3 and MPI-4 incorporate some of the research ideas
  - MPI implementations moving ahead with newer ideas for Exascale
  - Several optimizations inside implementations, and new functionality
- The work is not done, still a long way to go
  - But a start-from-scratch approach is neither practical nor necessary
  - Invest in orthogonal technologies that work with MPI (MPI+X)

Web: http://www.mcs.anl.gov/~balaji Email: balaji@anl.gov

Group website: http://www.mcs.anl.gov/group/pmrs/