Tools for Performance Debugging HPC Applications

David Skinner

deskinner@lbl.gov

U.S. DEPARTMENT OF ENERGY | Office of Science

NERSC National Energy Research Scientific Computing Center

BERKELEY LAB Lawrence Berkeley National Laboratory

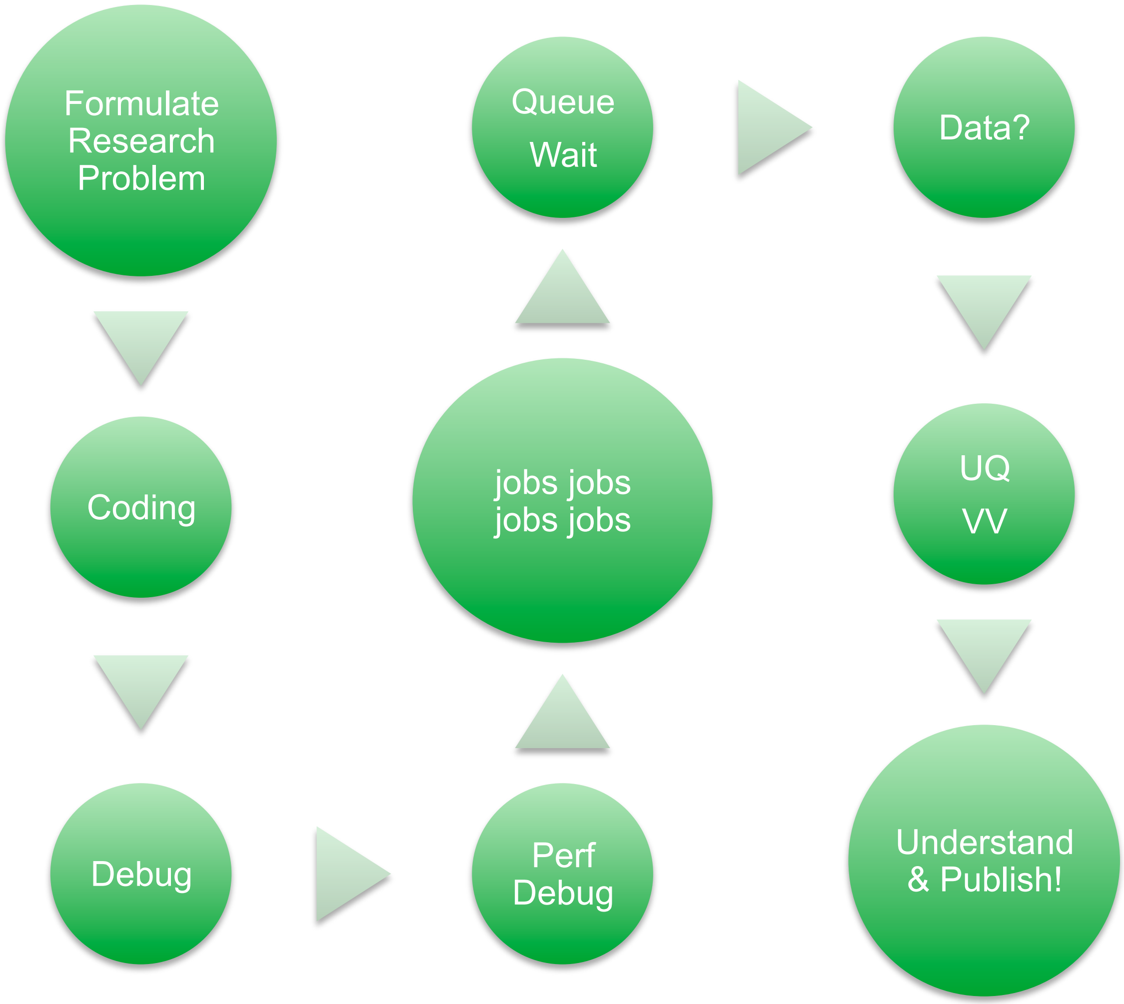
- **Practice**
  - Where to find tools
  - Specifics to NERSC and Hopper
- **Principles**
  - Topics in performance scalability
  - Examples of areas where tools can help
- **Scope & Audience**
  - Budding simulation scientist app dev
  - Compiler/middleware dev, YMMV

One Slide about NERSC

- **Serving all of DOE Office of Science**
  - domain breadth
  - range of scales
- **Science driven**
  - sustained performance
- **Lots of users**
  - ~4K active
  - ~500 logged in
  - ~300 projects
- **Architecture aware**
  - procurements driven by workload needs

Big Picture of Performance and Scalability

Performance is more than a single number

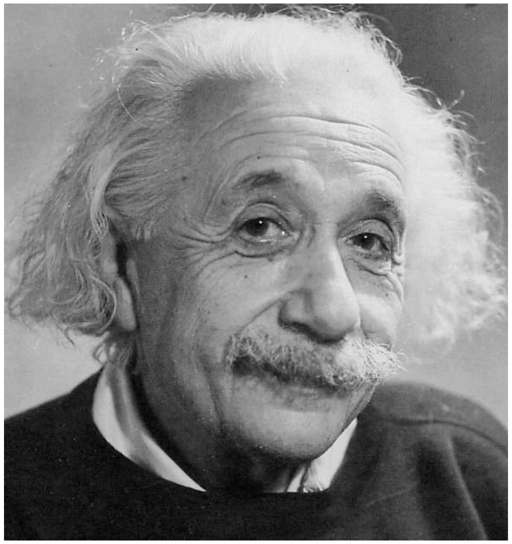


- •Plan where to put effort
- •Optimization in one area can de-optimize another
- •Timings come from timers and also from your calendar, time spent coding
- •Sometimes a slower algorithm is simpler to verify correctness

NERSC

Performance is Relative

- **To your goals**
  - Time to solution, $T_q + T_{wall}$ …
  - Your research agenda
  - Efficient use of allocation



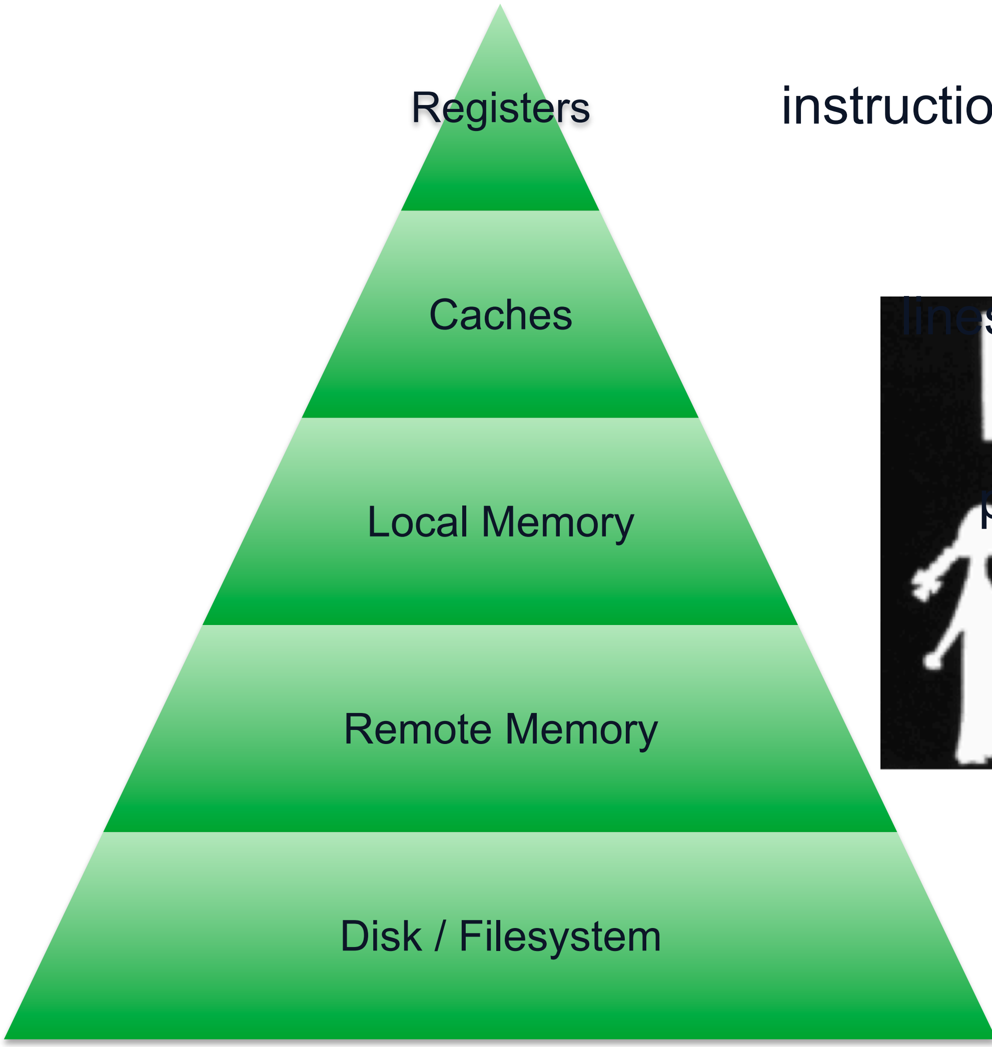
- **To the**
  - application code
  - input deck
  - machine type/state

Suggestion:
Focus on specific use cases as opposed to making *everything* perform well.
Bottlenecks can shift.

Specific Facets of Performance

- **Serial**
  - Leverage ILP on the processor
  - Feed the pipelines
  - Exploit data locality
  - Reuse data in cache
- **Parallel**
  - Expose concurrency
  - Minimizing latency effects
  - Maximizing work vs. communication

Performance is Hierarchical



instructions & operands

lines

pages

messages

blocks, files

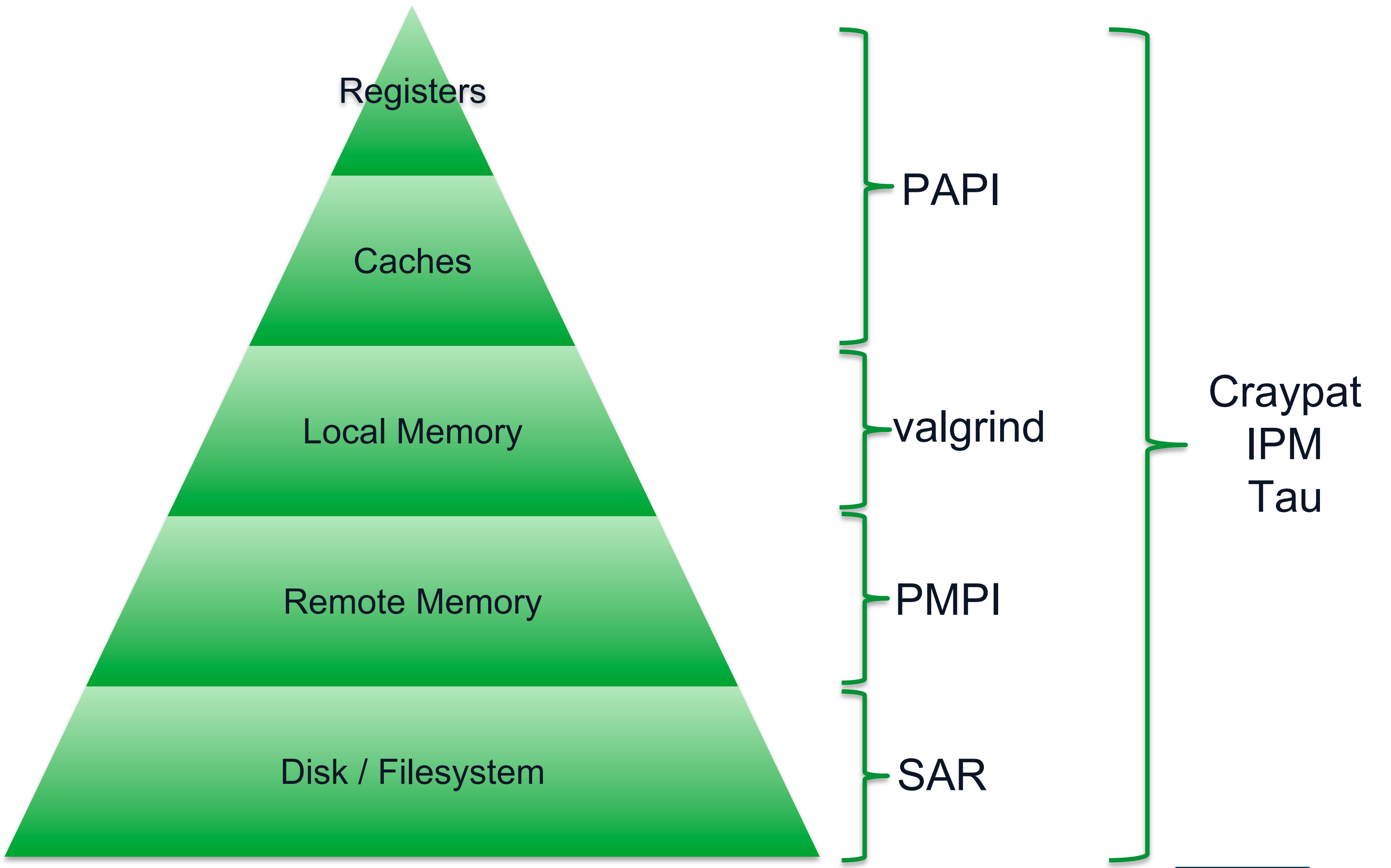
U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB Lawrence Berkeley National Laboratory

…on to specifics about HPC tools

Mostly at NERSC but fairly general

Tools are Hierarchical



HPC Perf Tool Mechanisms

- **Sampling**
  - Regularly interrupt the program and record where it is
  - Build up a statistical profile
- **Tracing / Instrumenting**
  - Insert hooks into program to record and time events
- **Use Hardware Event Counters**
  - Special registers count events on processor
  - E.g. floating point instructions
  - Many possible events
  - Only a few (~4 counters)

Typical Tool Use Requirements

- (Sometimes) Modify your code with macros, API calls, timers
- Compile your code
- Transform your binary for profiling/ tracing with a tool
- Run the transformed binary
  - A data file is produced
- Interpret the results with a tool

Performance Tools @ NERSC

- **Vendor Tools:**
  - CrayPat
- **Community Tools :**
  - TAU (U. Oregon via ACTS)
  - PAPI (Performance Application Programming Interface)
  - gprof
- **IPM: Integrated Performance Monitoring**

What HPC tools can tell us?

- **CPU and memory usage**
  - FLOP rate
  - Memory high water mark
- **OpenMP**
  - OMP overhead
  - OMP scalability (finding right # threads)
- **MPI**
  - % wall time in communication
  - Detecting load imbalance
  - Analyzing message sizes

Using the right tool

Tools can add overhead to code execution

- What level can you tolerate?

Tools can add overhead to scientists

- What level can you tolerate?

Scenarios:

- Debugging a code that is “slow”
- Detailed performance debugging
- Performance monitoring in production

Introduction to CrayPat

- **Suite of tools to provide a wide range of performance-related information**

- **Can be used for both sampling and tracing user codes**
  - with or without hardware or network performance counters
  - Built on PAPI

- **Supports Fortran, C, C++, UPC, MPI, Coarray Fortran, OpenMP, Pthreads, SHMEM**
- **Man pages**
  - intro_craypat(1), intro_app2(1), intro_papi(1)

Using CrayPat @ Hopper

1. **Access the tools**
   – `module load perftools`
2. **Build your application; keep .o files**
   – `make clean`
   – `make`
3. **Instrument application**
   – `pat_build ... a.out`
   – Result is a new file, `a.out+pat`
4. **Run instrumented application to get top time consuming routines**
   – `aprun ... a.out+pat`
   – Result is a new file XXXXX.xf (or a directory containing .xf files)
5. **Run pat_report on that new file; view results**
   – `pat_report XXXXX.xf > my_profile`
   – `vi my_profile`
   – Result is also a new file: XXXXX.ap2

Guidelines for Optimization

Derived metric	Optimization needed when*	PAT_RT_HWPC
Computational intensity	< 0.5 ops/ref	0, 1
L1 cache hit ratio	< 90%	0, 1, 2
L1 cache utilization (misses)	< 1 avg hit	0, 1, 2
L1+L2 cache hit ratio	< 92%	2
L1+L2 cache utilization (misses)	< 1 avg hit	2
TLB utilization	< 0.9 avg use	1
(FP Multiply / FP Ops) or (FP Add / FP Ops)	< 25%	5
Vectorization	< 1.5 for dp; 3 for sp	12 (13, 14)

* Suggested by Cray

Perf Debug and Production Tools

- **Integrated Performance Monitoring**
- **MPI profiling, hardware counter metrics, POSIX IO profiling**
- **IPM requires no code modification & no instrumented binary**
  - Only a “module load ipm” before running your program on systems that support dynamic libraries
  - Else link with the IPM library
- **IPM uses hooks already in the MPI library to intercept your MPI calls and wrap them with timers and counters**

IPM: Let's See

1) Do "module load ipm", link with $IPM, then run normally

2) Upon completion you get

```
##IPM2v0.xx##################################################
#
# command   : ./fish -n 10000
# start     : Tue Feb 08 11:05:21 2011   host      : nid06027
# stop      : Tue Feb 08 11:08:19 2011   wallclock : 177.71
# mpi_tasks : 25 on 2 nodes              %comm     : 1.62
# mem [GB]  : 0.24                       gflop/sec : 5.06
…
```

Maybe that's enough. If so you're done.

Have a nice day ☺

IPM : IPM_PROFILE=full

```
# host    : s05601/006035314C00_AIX        mpi_tasks : 32 on 2 nodes
# start   : 11/30/04/14:35:34              wallclock : 29.975184 sec
# stop    : 11/30/04/14:36:00              %comm     : 27.72
# gbytes  : 6.65863e-01 total              gflop/sec : 2.33478e+00 total
#                           [total]        <avg>          min           max
# wallclock                 953.272        29.7897        29.6092       29.9752
# user                      837.25         26.1641        25.71         26.92
# system                    60.6           1.89375        1.52          2.59
# mpi                       264.267        8.25834        7.73025       8.70985
# %comm                                    27.7234        25.8873       29.3705
# gflop/sec                 2.33478        0.0729619      0.072204      0.0745817
# gbytes                    0.665863       0.0208082      0.0195503     0.0237541
# PM_FPU0_CMPL              2.28827e+10    7.15084e+08    7.07373e+08   7.30171e+08
# PM_FPU1_CMPL              1.70657e+10    5.33304e+08    5.28487e+08   5.42882e+08
# PM_FPU_FMA                3.00371e+10    9.3866e+08     9.27762e+08   9.62547e+08
# PM_INST_CMPL              2.78819e+11    8.71309e+09    8.20981e+09   9.21761e+09
# PM_LD_CMPL                1.25478e+11    3.92118e+09    3.74541e+09   4.11658e+09
# PM_ST_CMPL                7.45961e+10    2.33113e+09    2.21164e+09   2.46327e+09
# PM_TLB_MISS               2.45894e+08    7.68418e+06    6.98733e+06   2.05724e+07
# PM_CYC                    3.0575e+11     9.55467e+09    9.36585e+09   9.62227e+09
#                           [time]         [calls]        <%mpi>        <%wall>
# MPI_Send                  188.386        639616         71.29         19.76
# MPI_Wait                  69.5032        639616         26.30         7.29
# MPI_Irecv                 6.34936        639616         2.40          0.67
# MPI_Barrier               0.0177442      32             0.01          0.00
# MPI_Reduce                0.00540609     32             0.00          0.00
# MPI_Comm_rank             0.00465156     32             0.00          0.00
# MPI_Comm_size             0.000145341    32             0.00          0.00
```

Advice: Develop (some) portable approaches to performance

- **There is a tradeoff between vendor-specific and vendor neutral tools**
  - Each have their roles, vendor tools can often dive deeper
- **Portable approaches allow apples-to-apples comparisons**
  - Events, counters, metrics may be incomparable across vendors
- **You can find printf most places**
  - Put a few timers in your code?

Examples of HPC tool usage

Scaling: definitions

- **Scaling studies involve changing the degree of parallelism. Will we be change the problem also?**
- **Strong scaling**
  - Fixed problem size
- **Weak scaling**
  - Problem size grows with additional resources
- **Speed up = $T_s/T_p(n)$**
- **Efficiency = $T_s/(n*T_p(n))$**

Be aware there are multiple definitions for these terms

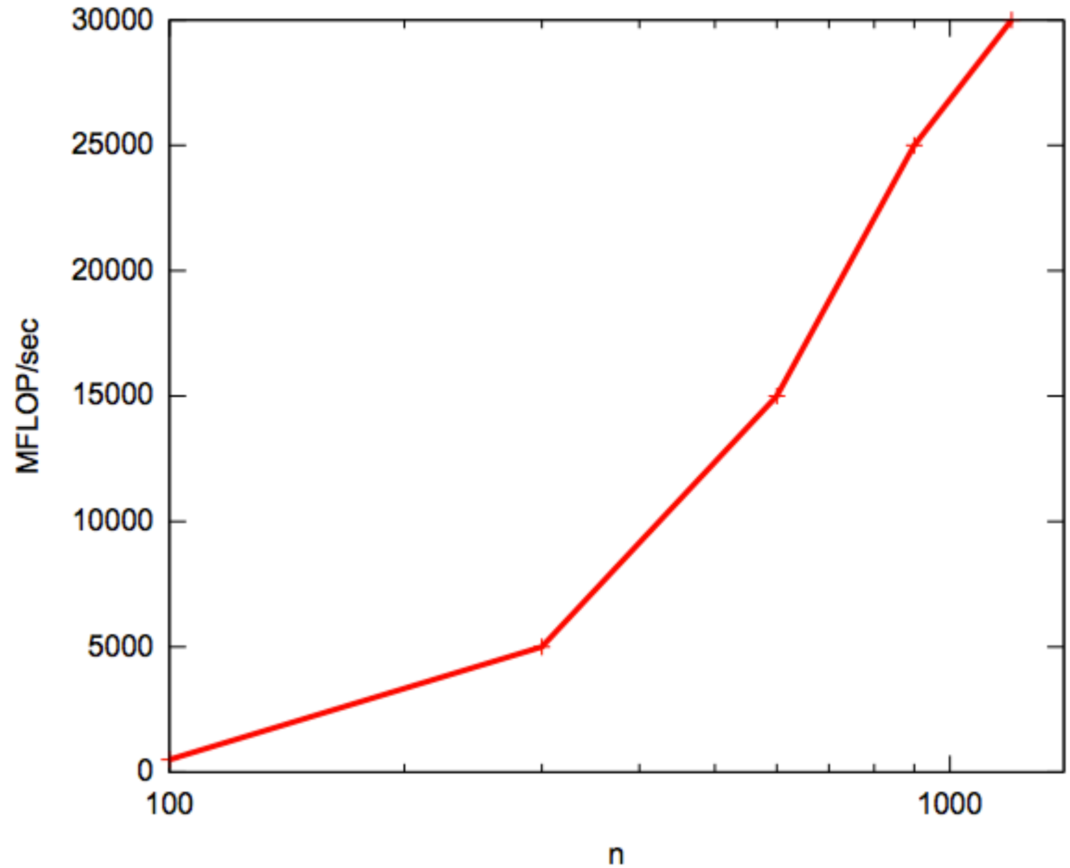
Conducting a scaling study

With a particular goal in mind, we systematically vary concurrency and/or problem size

Example:

How large a 3D (n^3) FFT can I efficiently run on 1024 cpus?

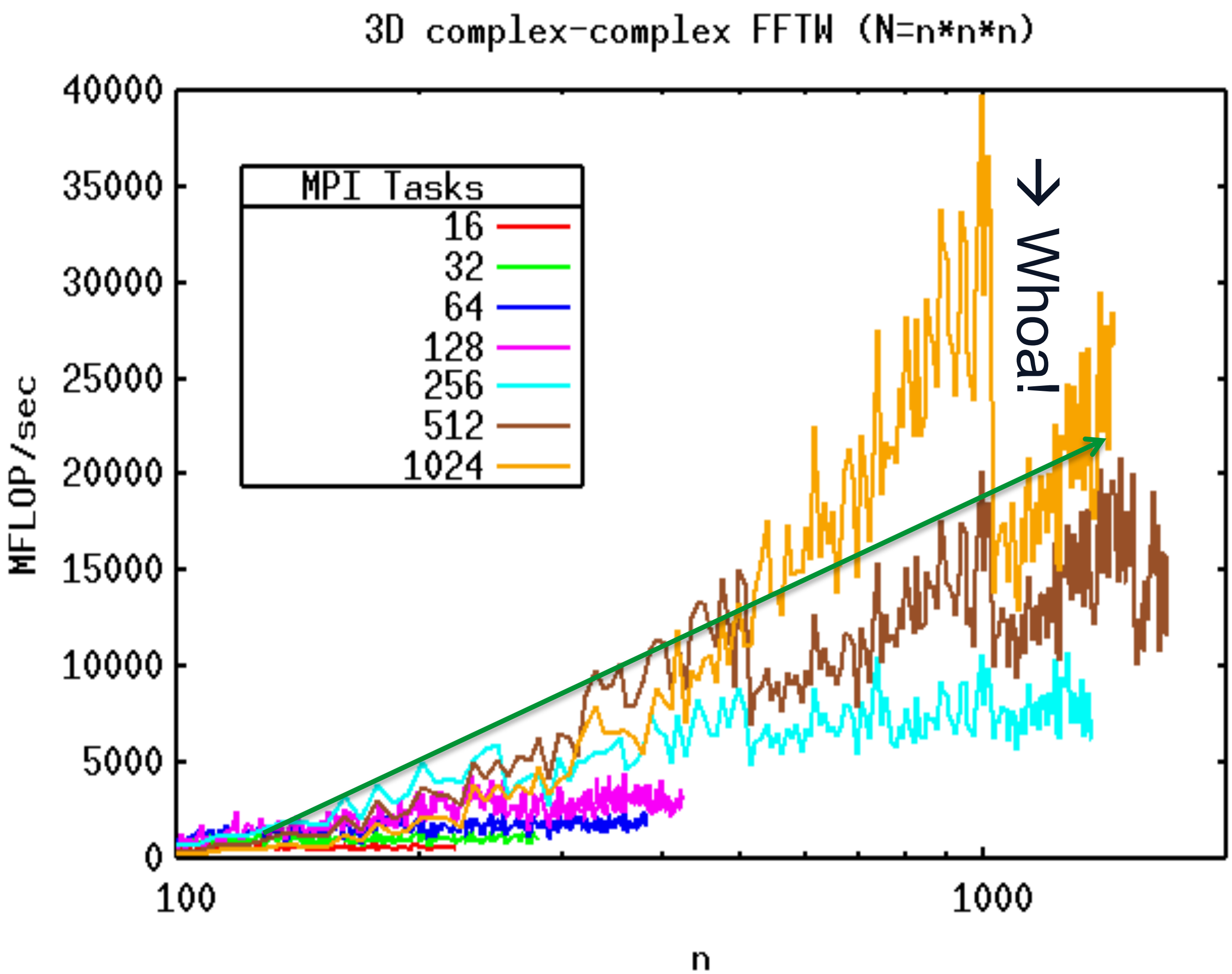
Looks good?



U.S. DEPARTMENT OF ENERGY | Office of Science

Let’s look a little deeper….

The scalability landscape

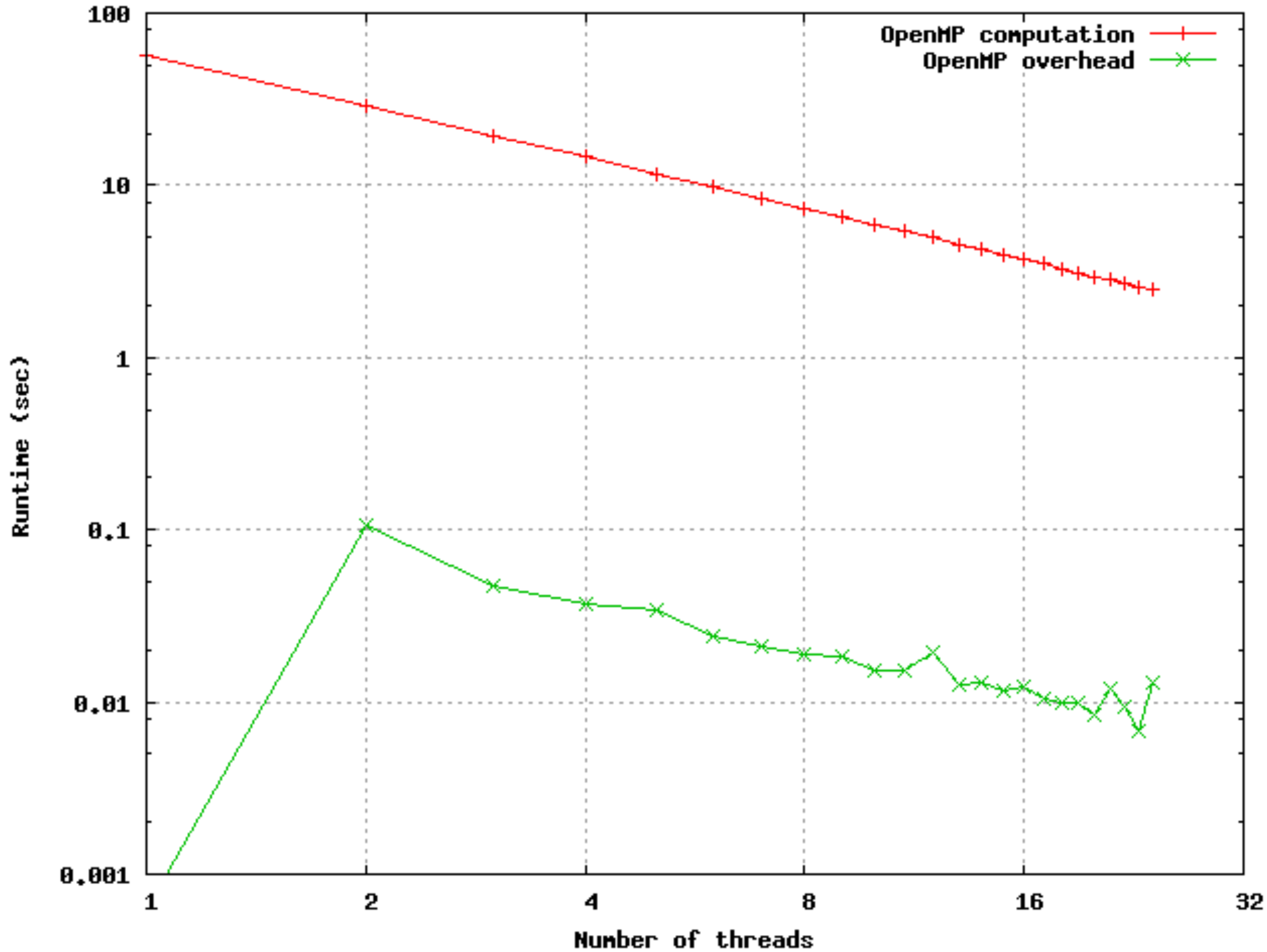


Why so bumpy?

- Algorithm complexity or switching
- Communication protocol switching
- Inter-job contention
- ~bugs in vendor software

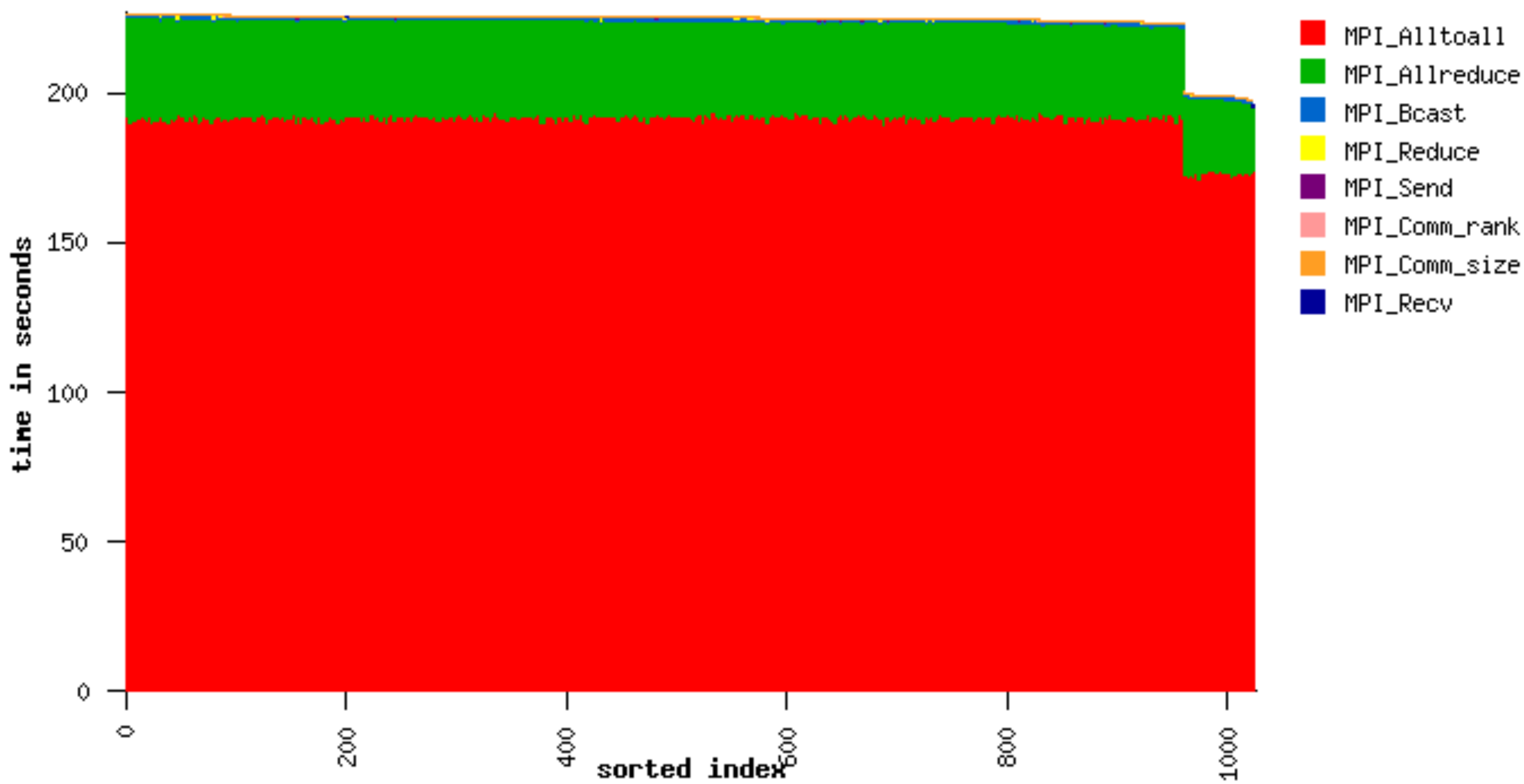
Not always so tricky

Main loop in jacobi_omp.f90; ngrid=6144 and maxiter=20



Load Imbalance : Pitfall 101

Communication Time: 64 tasks show 200s, 960 tasks show 230s

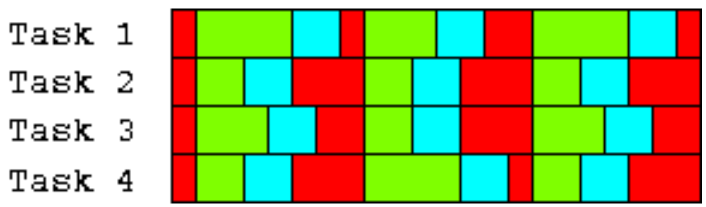


MPI ranks sorted by total communication time

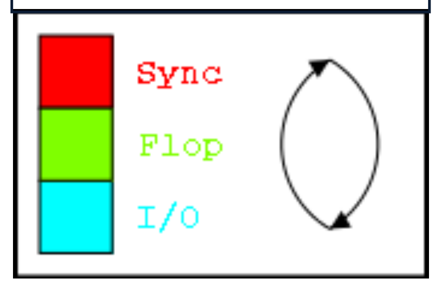
U.S. DEPARTMENT OF ENERGY | Office of Science

Load Balance : cartoon

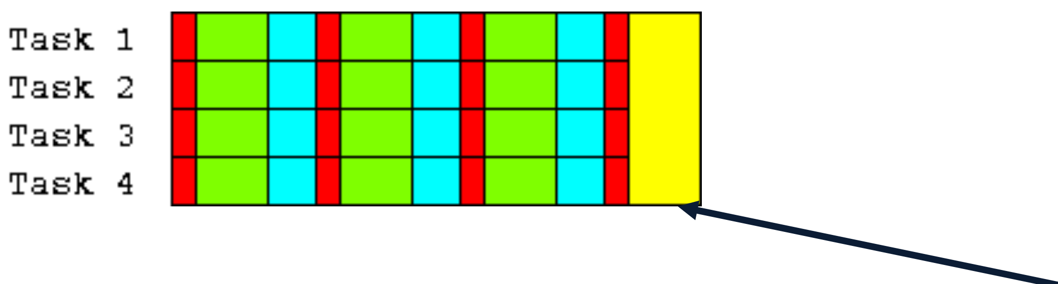
Unbalanced:



Universal App

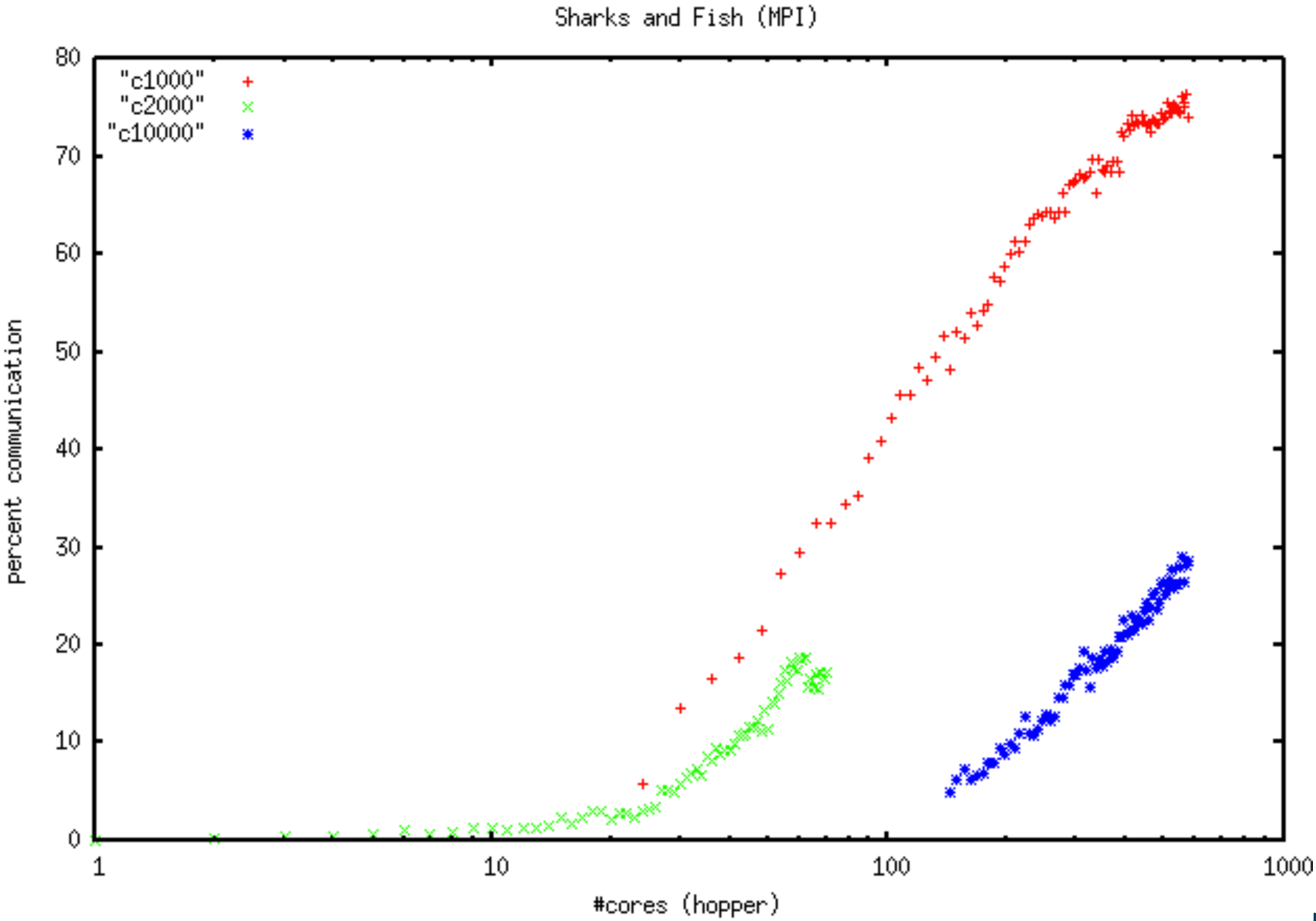


Balanced:



Time saved by load balance

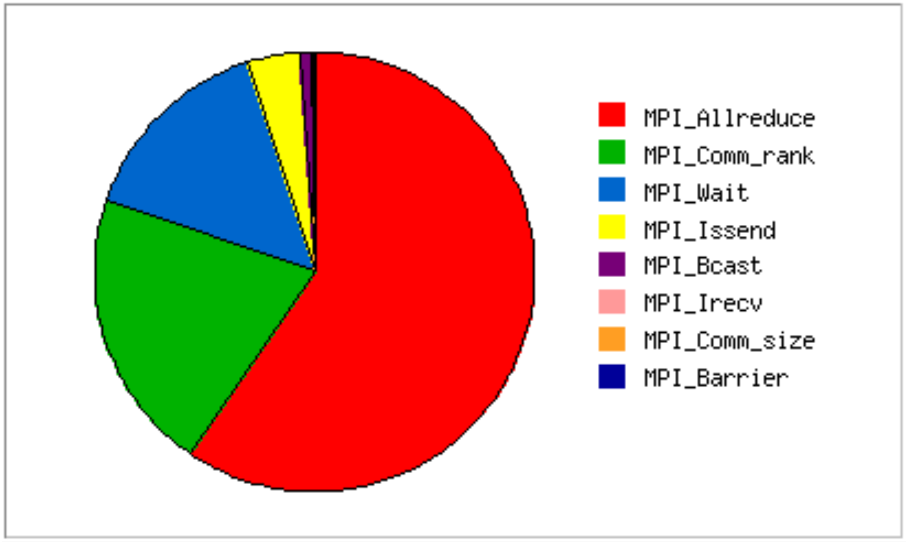
Too much communication



Simple Stuff: What's wrong here?

Communication

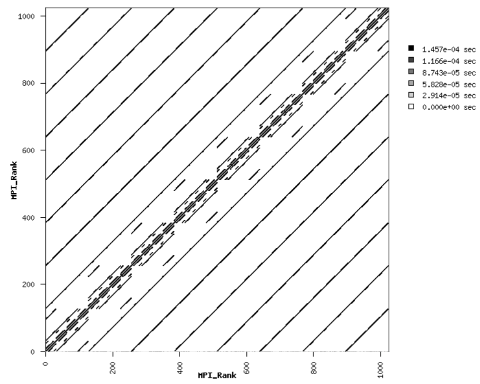
% of MPI Time



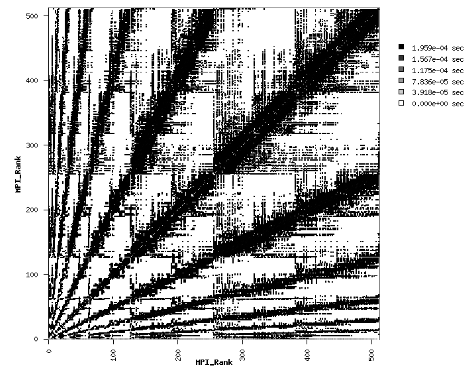
Communication Event Statistics (100.00% detail)

	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Allreduce	8	3278848	124132.547	0.000	114.920	59.35	16.88
MPI_Comm_rank	0	35173439489	43439.102	0.000	41.961	20.77	5.91
MPI_Wait	98304	13221888	15710.953	0.000	3.586	7.51	2.14
MPI_Wait	196608	13221888	5331.236	0.000	5.716	2.55	0.72
MPI_Wait	589824	206848	5166.272	0.000	7.265	2.47	0.70

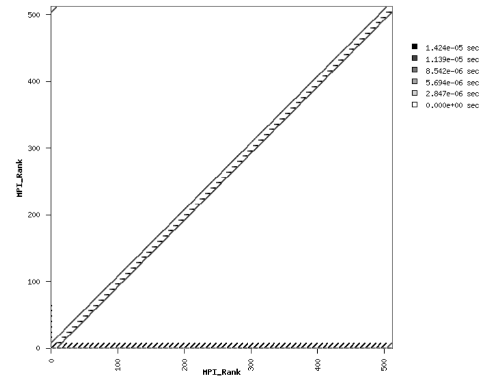
Not so simple: Comm. topology



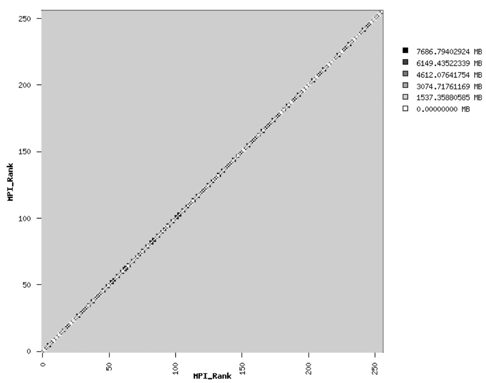
MILC



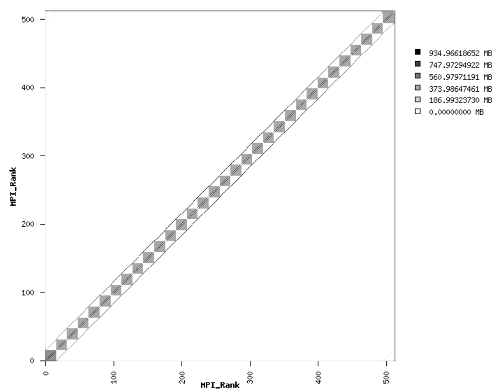
MAESTRO



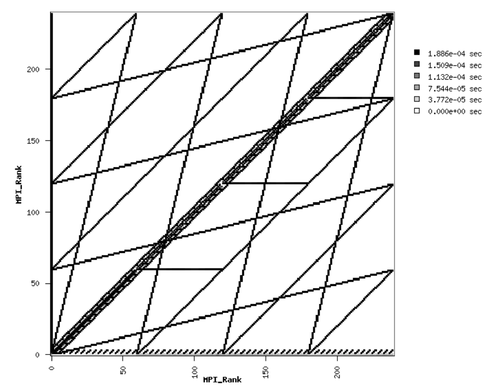
GTC



PARATEC



IMPACT-T



CAM

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Performance in Batch Queue Space

A few notes on queue optimization

Consider your schedule

- **Charge factor**
  - regular vs. low
- **Scavenger queues**
- **Xfer queues**
  - Downshift concurrency

Consider the queue constraints

- **Run limit**
- **Queue limit**
- **Wall limit**
  - Soft (can you checkpoint?)

Jobs can submit other jobs

Marshalling your own workflow

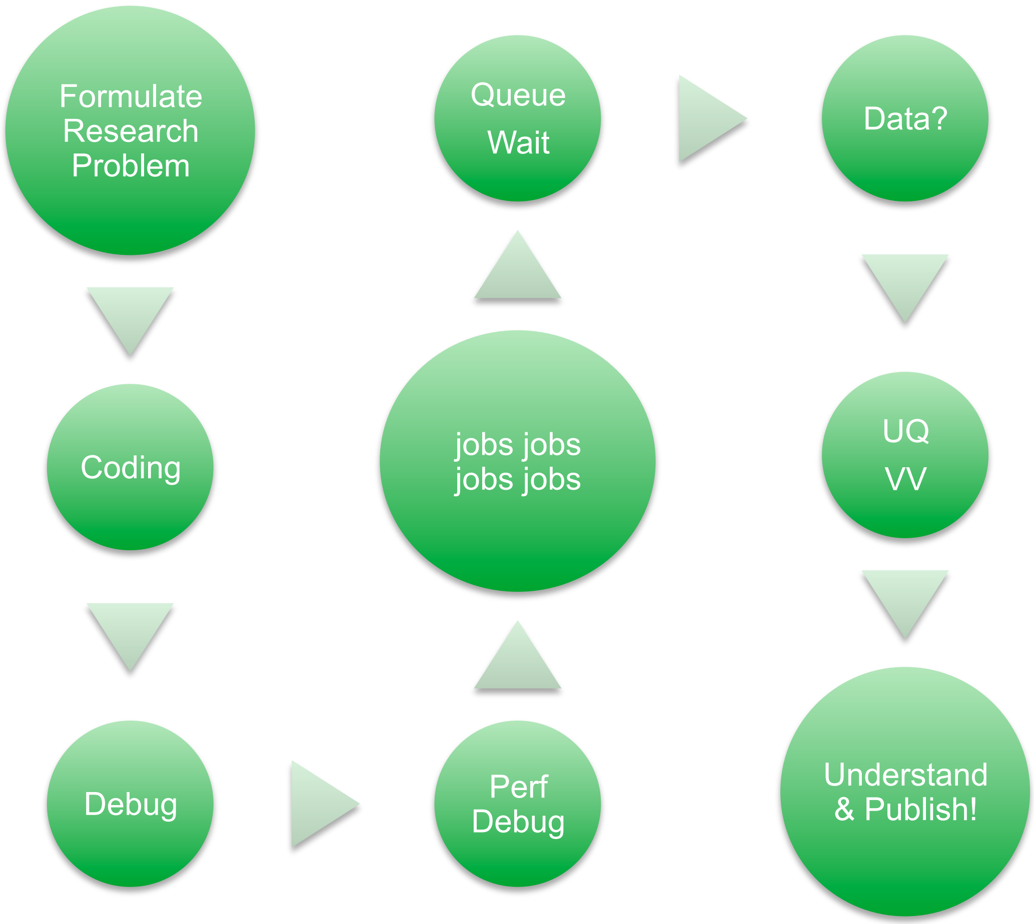
- **Lots of choices in general**
  - Hadoop, CondorG, MySGE
- **On hopper it's easy**

```
#PBS -l mppwidth=4096
aprun –n 512 ./cmd &
aprun –n 512 ./cmd &
…
aprun –n 512 ./cmd &

wait
```

```
#PBS -l mppwidth=4096
while(work_left) {
 if(nodes_avail) {
 aprun –n X next_job &
 }
wait
}
```

Thanks!



Contacts:

help@nersc.gov

deskinner@lbl.gov