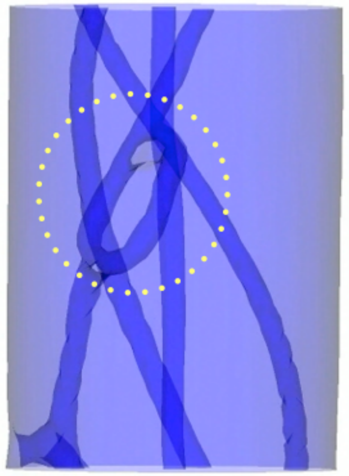
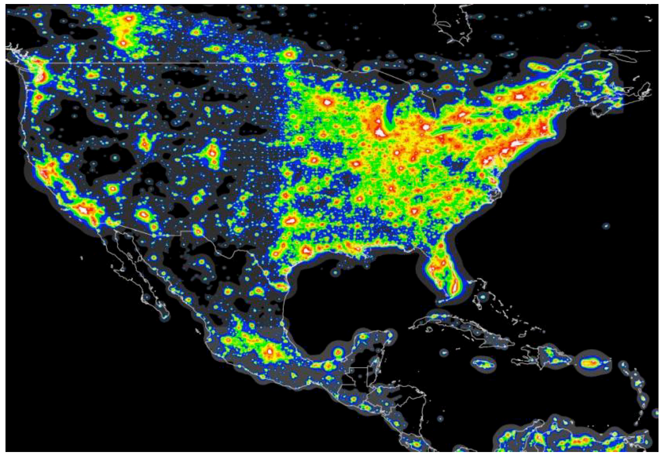


quantum vortex generation in UFG (ref M. Forbes lectures on SLDA in UFG)



energy use by population distribution

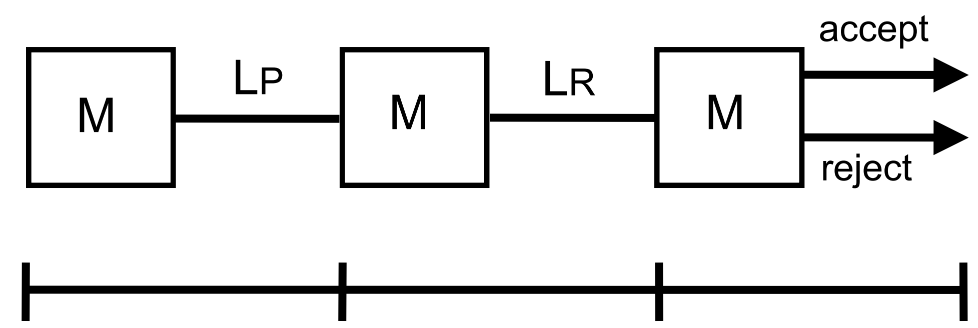
INT Summer School: Computing

*calibrated according to computing survey results

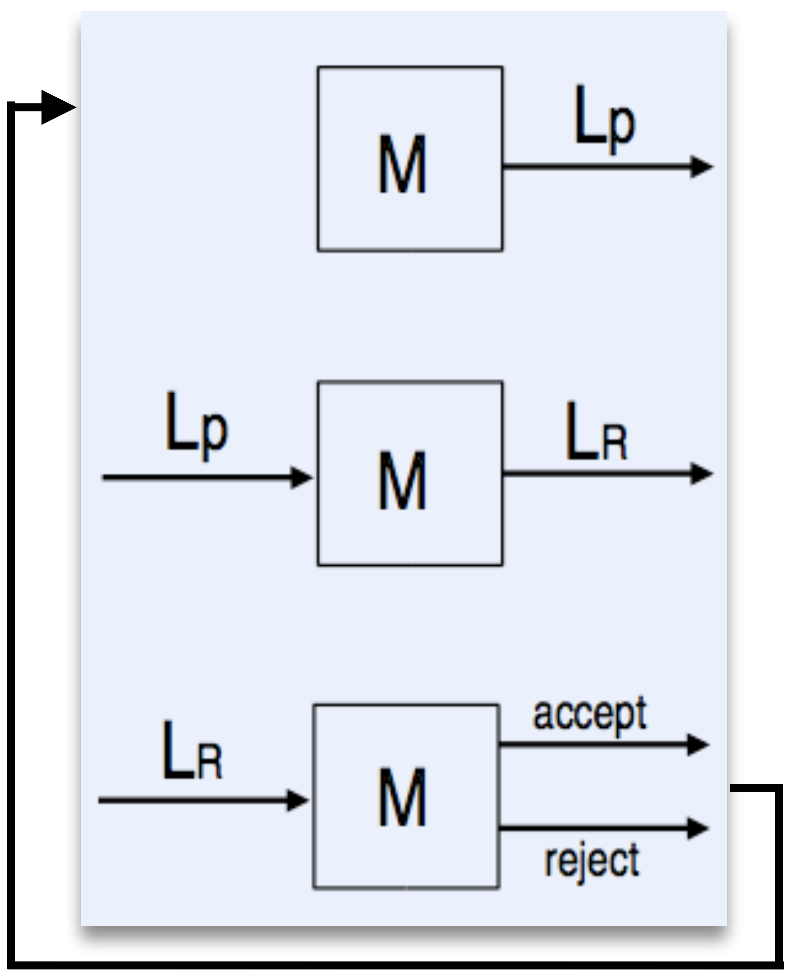
K. J. Roche
High Performance Computing Group, Pacific Northwest National Laboratory
Nuclear Theory Group, University of Washington

Concepts

- COMPLEXITY
  - PROBLEMS
  - ALGORITHMS
  - MACHINES



Measured time for machine M to generate the language of the problem plus time to generate the language of the result plus the time to accept or reject the language of the result.



Asking questions, solving problems is recursive process

Accepting a result means a related set of conditions is satisfied

$$S = S1 \wedge S2 \wedge ... \wedge Sn$$

algorithm, a Turing machine that always halts
decidable problems are posed as a recursive language
undecidable problems have no algorithms that accept the language of the problem and generate / accept or reject an answer (Rice's Theorem posits that non-trivial properties of r.e. languages are undecidable. Examples are emptiness, finiteness, regularity, and context freedom.)

let's be practical (we only have 1 hour)

CPU	**M**
Memory	
Network	
External Devices ...	

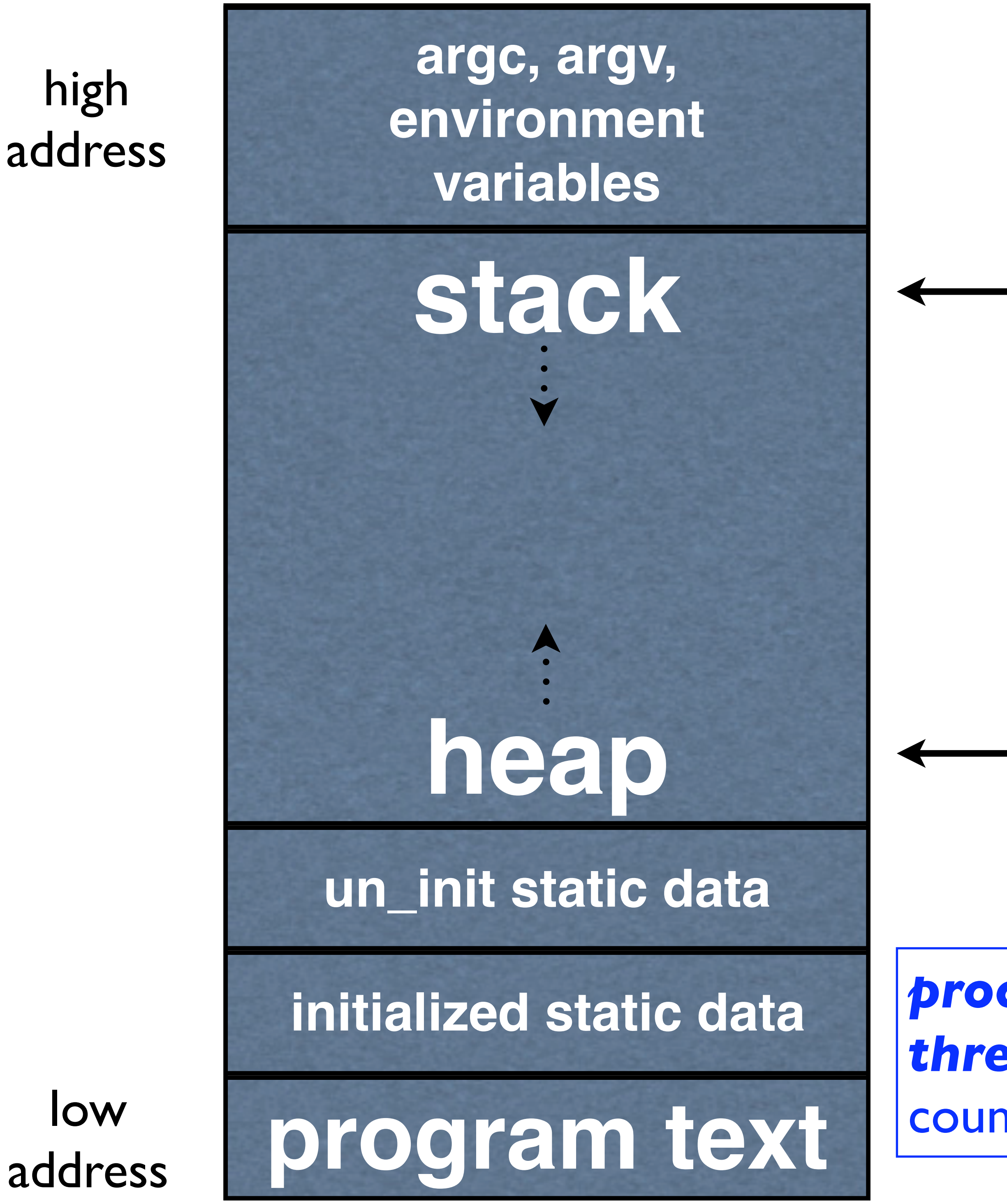
CPU

- **ALU**, adds, comparisons
- **FPU**, floating point operations
- **L/S U**, data loads / stores
- **Registers**, fast memory; FPR, GPR, etc.
- **PC**, program counter -address in memory of instruction that is executing (control flow, fetch / decode in CPU)
- **Memory interface**, often L1 and L2 caches

other:
clock speed
buses
ISA (Intel x86 most popular, x86-64, ...)

Memory

- storage for active data and programs



activation records for function calls (return adresses, parameters, saved registers, automatice variables)

memory allocations (malloc)

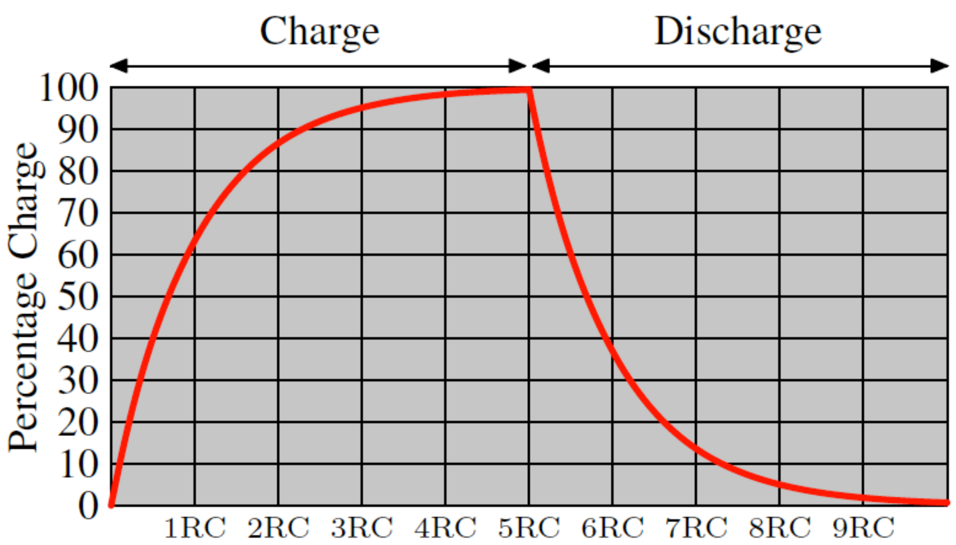
process, a program in execution -has this structure
thread, an abstract data type w/ its own stack, program counter value, register set, and state

model of program placement in memory

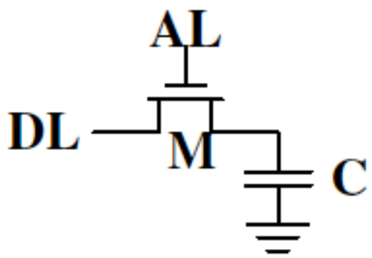
Memory

- size
  - virtual memory (looks bigger than it is)
- hierarchy
  - try to improve performance by reducing latency
- bandwidth
- correction mechanisms
- WHAT ABOUT COST / PERFORMANCE?!

Power = Capacity * Voltage^2 * Frequency



DRAM
C, capacitor, keeps cell state



M, transistor, controls access to cell state

read the state of the cell the **access line AL** is raised
-causes a current to flow on the data line DL or not

write to the cell the **data line DL** is appropriately set and AL is raised for a time long enough to charge or drain the capacitor

Today's Memories ...

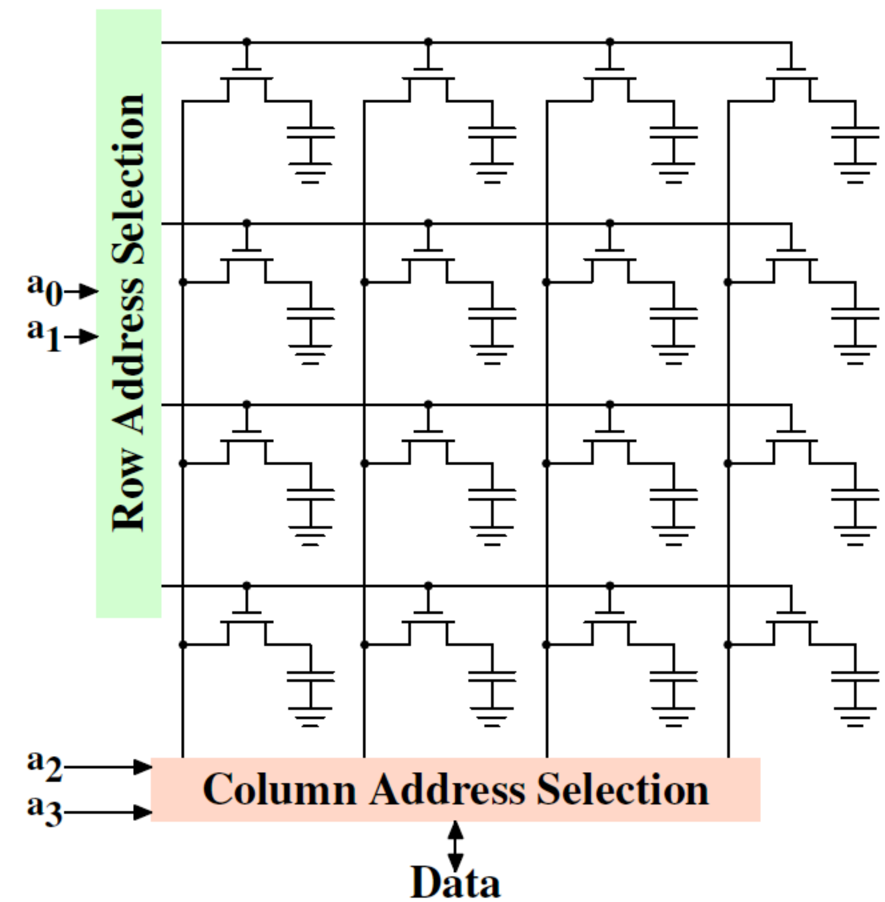
- 10^9 cells
- cell capacitance < femto-farad
- resistance O(tera-ohms)

Refresh Cycles ~ 64ms

- leakage
- reading drains the charge (read + recharge)

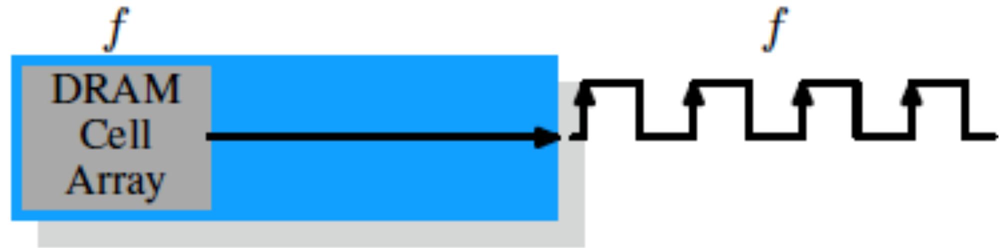
Faster memory

- lower voltage --> decreases stability,
- increase frequency --> $$$ as arrays get large
- (i.e. more addressable memory) and voltage is increased to assure stability

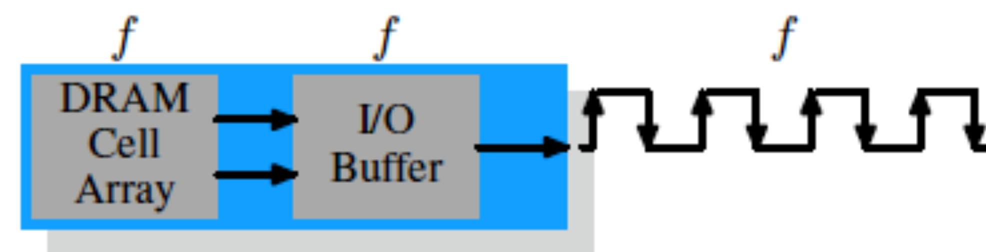


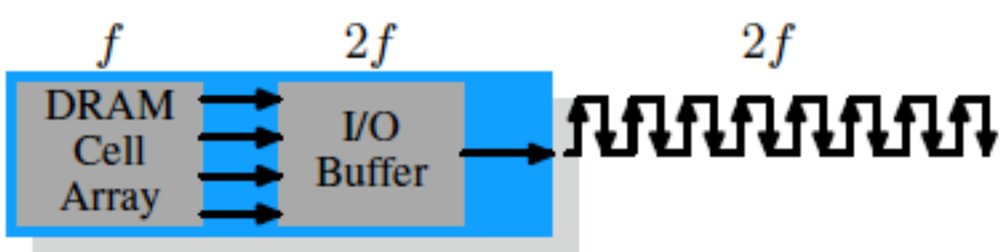
ref. Drepper, What every Programmer Should Know about Memory

SDR (PC100) ~
DRAM cell array 100MHz
data transfer rate 100Mbps



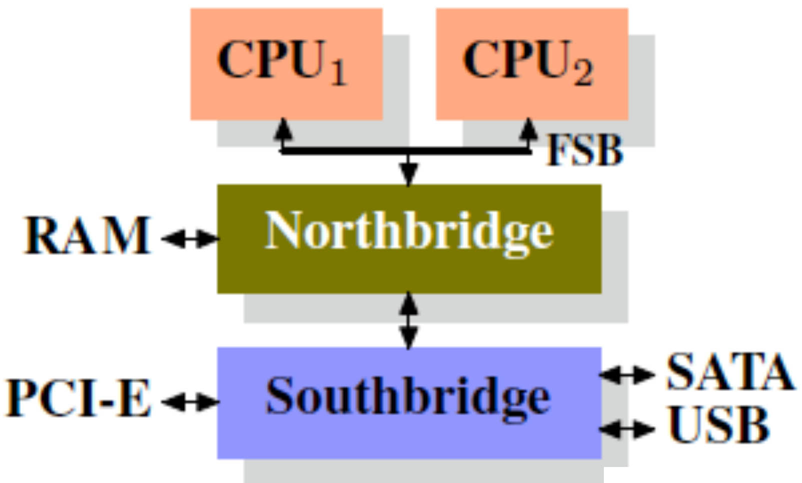
DDR (PC1600) ~ moves 2X the data / clock (leading , falling)
add "I/O" buffer (2 bits on data line) adjacent to DRAM cell array
pull two adjacent column cells per access over 2 line data bus
100 MHz X 64 bit / data bus X 2 data bus lines = 1600 MBps

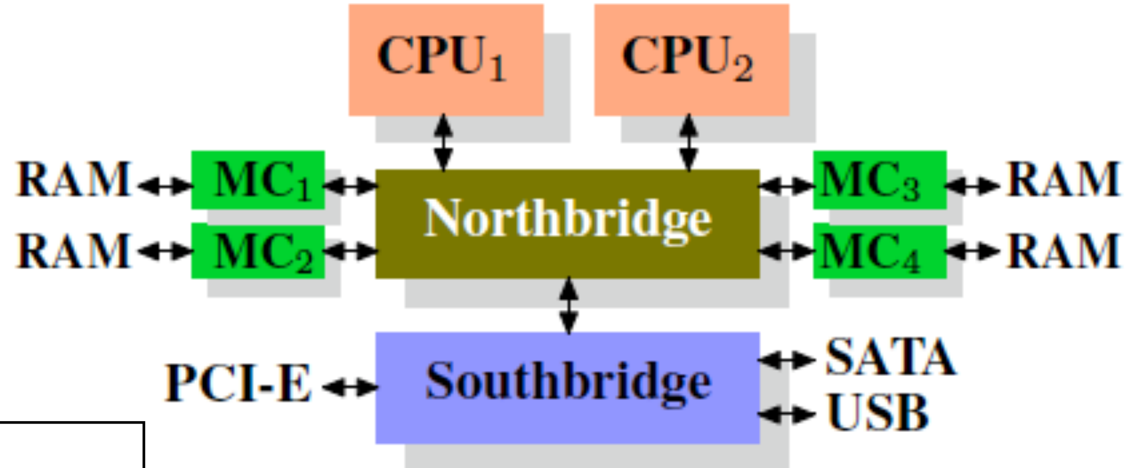




DDR2 (PC6400) ~ moves 4X the data / clock
double the bus frequency --> 2X bandwidth
double "I/O" buffer speed to match the bus
4 bits / clock on 4 line data bus
200MHz array; 400MHz bus; 800MHz FSB (effective freq)
200 MHz X 64 bit / data bus X 4 data bus lines = 6400 MBps
240 PIN addressing @ 1.8V

***each stall cycle on the memory bus is > 11 cpu cycles even in the best systems**

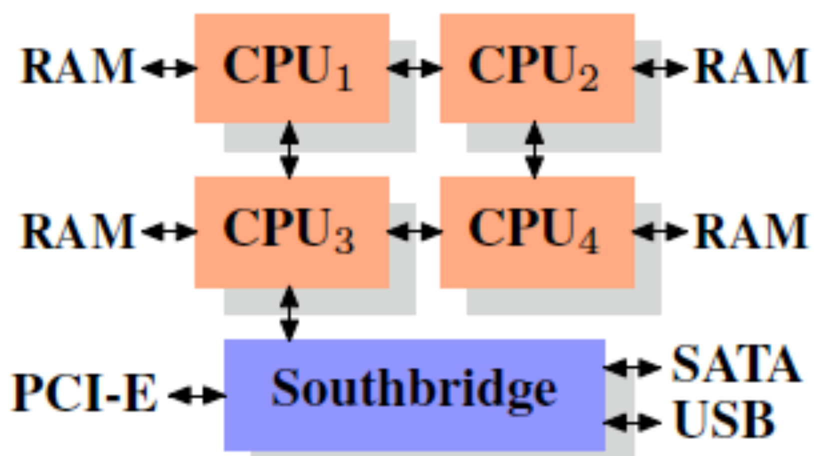




external memory control

* 2X for both lanes

PCI := 132 MB/s
AGP 8X := 2,100 MB/s
PCI Express 1x := 250 [500]* MB/s
PCI Express 2x := 500 [1000]* MB/s
PCI Express 4x := 1000 [2000]* MB/s
PCI Express 8x := 2000 [4000]* MB/s
PCI Express 16x := 4000 [8000]* MB/s
PCI Express 32x := 8000 [16000]* MB/s
USB 2.0 (Max Possible) := 60 MB/s
IDE (ATA100) := 100 MB/s
IDE (ATA133) := 133 MB/s
SATA := 150 MB/s
SATA II := 300 MB/s
Gigabit Ethernet := 125 MB/s
IEEE1394B [Firewire 800] := ~100 MB/s



cpu integrated memory control

Prototypical Computing Platforms: Yesterday

Hex-Core AMD Opteron (TM)	2.6e9 Hz clock	4 FP_OPs / cycle / core 128 bit registers
PEs	18,688 nodes	224,256 cpu-cores (processors)
Memory	16 GB / node 6 MB shared L3 / chip 512 KB L2 / core 64 KB D,I L1 / core	dual socket nodes 800 MHz DDR2 DIMM 25.6 GBps / node memory bw
Network	AMD HT SeaStar2+	3D torus topology 6 switch ports / SeaStar2+ chip 9.6 GBps interconnect bw / port 3.2GBps injection bw
Operating Systems	Cray Linux Environment (CLE) (xt-os2.2.41A)	SuSE Linux on service / io nodes

FY	Aggregated Cycles	Aggregated Memory	Aggregated FLOPs	Memory/FLOPs
2008	65.7888 THz	61.1875 TB	263.155 TF	0.2556
2009	343.8592 THz	321.057 TB	1.375 PF	0.2567
2010 / 11	583.0656 THz	321.057 TB	2.332 PF	0.1513

Measurements

- application specific measures / metrics (see bonus for examples)
- machine events
  - clear dependence on tools / hardware support to monitor hardware components activated during program execution
    - --cycle count, disk accesses, floating point operation counts, instructions issued and retired, L2 data cache misses, maximum memory set size, number of loads / stores etc.
  - derived measures
    - efficiency, cycles per instruction (CPI) or floating point operations retired per second (FLOPs)
    - computational costs, CPU Hours (relates execution time to processing elements), etc.
  - pinpoint insufficient parallelism, lock contention, and parallel overheads in threading and synchronization strategies

Enhancement Modes

- *performance* (improve efficiency, scalability - weak or strong)
  - -data structures / discretizations, algorithms, libraries, language enhancements, compilers
- *scientific* (better accuracy, improved predictive power)
  - -physical models, the problem representation, validity of inputs, and correctness of computed results

Strong Scaling

Machine Events	Q2	Q4
INS	2.147E+15	2.1130E+15
FP_OP	5.896E+14	5.8947E+14
PEs	5632	11264
Time[s]	121.252233	57.222988

INS:
2113046508030116 /
2146627269408190 = **.9843**

FP_OP:
589469277576687 /
589624961638025 = **.9997**

PEs: 11264 / 5632 = **2**

Time[s]:
57.222988 / 121.252233 = **.472**

Weak Scaling

Machine Events	Q2	Q4
INS	5.18E+17	1.93E+18
FP_OP	4.63E+17	1.81E+18
PEs	7808	31232
Time[s]	25339	23791

INS: 3.72

FP_OP: 3.92

PEs: 4

Time[s]: .938

NB: k= T(Q4)*PEs(Q4)/
T(Q2)*PEs(Q2) ~ 3.756

Improve Efficiency

Machine Events	Q2	Q4
INS	3.16E+12	4.37E+11
FP_OP	5.50E+11	5.53E+11
PEs	1	1
L2DCM	823458808	34722900
Time[s]	826.494142	79.414198

INS: 0.1381 (7.239x)

FP_OP: 1.0053 (0.99475x)

PEs: 1

L2DCM: 0.0422 (23.715x)

Time[s]: 0.0961 (10.407x)

"simulating the same problem in less time"

Algorithm, machine strong scaling :

Q4 problem := Q2 problem
Q4 algorithm := Q2 algorithm
Q4 machine ~ k * Q2 machine
Q4 time ~ 1/k * Q2 time

Algorithm enhancements, performance optimizations:

Q4 problem := Q2 problem
Q4 algorithm ~ enhanced Q2 algorithm
Q4 machine := Q2 machine
Q4 time ~ 1/k * Q2 time

*Could consider other variations: algorithm and machine are varied to achieve reduction of compute time

"simulating a larger problem in same time"

Algorithm, machine weak scaling (100%):

Q4 problem ~ k * Q2 problem
Q4 algorithm := Q2 algorithm
Q4 machine ~ k * Q2 machine
Q4 time := Q2 time

Algorithm enhancements, performance optimizations:

Q4 problem ~ k * Q2 problem
Q4 algorithm ~ enhanced Q2 algorithm
Q4 machine := Q2 machine
Q4 time := Q2 time

*Could consider other variations: problem, algorithm and the machine are varied to achieve fixed time assertion

Computational Efficiency

- Total elapsed time to execute a problem instance with a specific software instance (algorithm) on a machine instance
- Parallel
  - e(n,p) := Tseq (n) / (p * T(n,p))

weighted:
(t*nPEs/DOF)_b/(t*nPEs/DOF)_e

Data Structures

linked lists
queues
graphs
tensors
lattices , meshes
stencils (for PDEs)
...

Graphs

- G(V,E)
  - V, vertex set, |V| cardinality
  - E, edge set, (vi,vj),..., |E|
  - values on vertices
  - values on edges

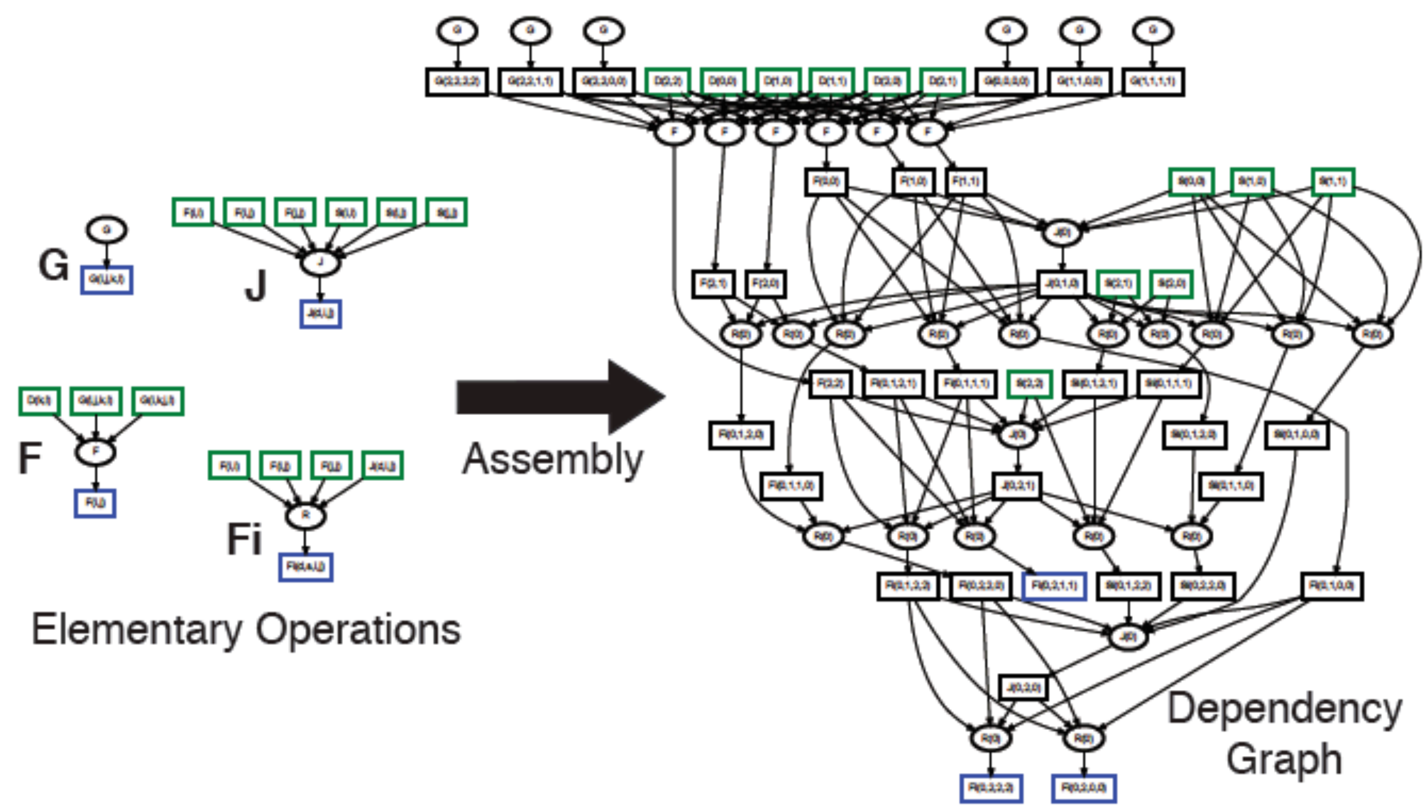
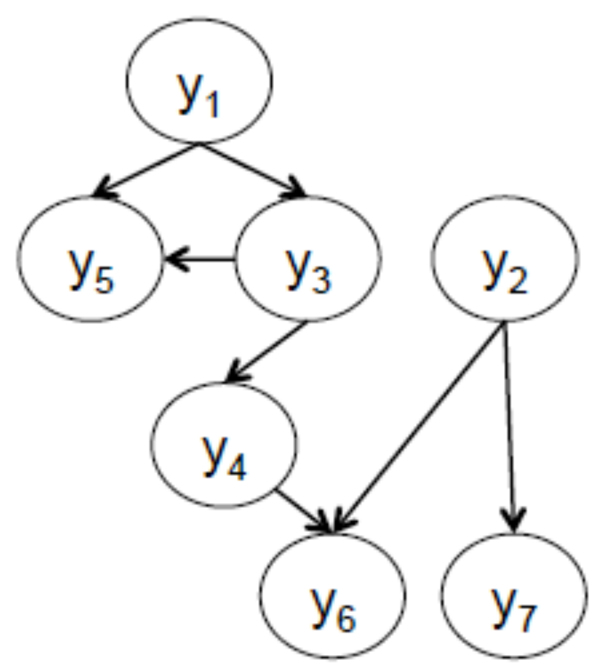


Figure 1. Dependency graph for a portion of the Hartree-Fock procedure.

L

l_{11}								y_1		r_1
	l_{22}							y_2		r_2
l_{31}		l_{33}						y_3		r_3
		l_{43}	l_{44}				*	y_4	=	r_4
l_{51}		l_{53}		l_{55}				y_5		r_5
	l_{62}		l_{64}		l_{66}			y_6		r_6
	l_{72}					l_{77}		y_7		r_7



Sparse Matrices

- basic for NK based methods that execute repeated SpMV accumulate operations
- 2 integer arrays, 1 value array (i.e. double precision numbers)

$$A = \begin{array}{c} \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix} \\ \begin{pmatrix} 1 & * & * & 2 & * & * & * & * & * & 3 \\ * & 4 & 5 & * & 6 & * & * & 7 & * & * \\ * & * & 8 & 9 & * & * & 10 & * & * & * \\ * & 11 & * & 12 & * & * & * & * & * & * \\ 13 & * & * & 14 & 15 & * & * & * & * & 16 \\ * & * & * & 17 & * & 18 & * & * & * & * \\ * & * & 19 & * & 20 & * & 21 & * & * & * \\ * & * & * & 22 & * & * & * & 23 & 24 & * \\ 25 & * & * & * & * & 26 & * & * & 27 & * \\ * & 28 & * & * & * & * & * & * & * & 29 \end{pmatrix} \end{array}$$

$$\alpha = \begin{pmatrix} 3 \\ 4 \\ 3 \\ 2 \\ 4 \\ 2 \\ 3 \\ 3 \\ 3 \\ 2 \end{pmatrix}, \beta = \begin{pmatrix} 0 & 3 & 9 & \\ 1 & 2 & 4 & 7 \\ 2 & 3 & 6 & \\ 1 & 3 & & \\ 0 & 3 & 4 & 9 \\ 3 & 5 & & \\ 2 & 4 & 6 & \\ 3 & 7 & 8 & \\ 0 & 5 & 8 & \\ 1 & 9 & & \end{pmatrix}, \hat{A} = \begin{pmatrix} 1 & 2 & 3 & \cdots & 29 \end{pmatrix}^T$$

- number of nonzeros in row
- column index
- values

Dense Matrices

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} = \begin{pmatrix} (0+0*4=0) & (0+1*4=4) & (0+2*4=8) & (0+3*4=12) \\ (1+0*4=1) & (1+1*4=5) & (1+2*4=9) & (1+3*4=13) \\ (2+0*4=2) & (2+1*4=6) & (2+2*4=10) & (2+3*4=14) \\ (3+0*4=3) & (3+1*4=7) & (3+2*4=11) & (3+3*4=15) \end{pmatrix}$$

$$A = \begin{pmatrix} 0 & 7 & 14 & 21 & 28 & 35 & 42 \\ 1 & 8 & 15 & 22 & 29 & 36 & 43 \\ 2 & 9 & 16 & 23 & 30 & 37 & 44 \\ 3 & 10 & 17 & 24 & 31 & 38 & 45 \\ 4 & 11 & 18 & 25 & 32 & 39 & 46 \\ 5 & 12 & 19 & 26 & 33 & 40 & 47 \\ 6 & 13 & 20 & 27 & 34 & 41 & 48 \end{pmatrix}$$

$$f(2d\_block\_cyclic) \longrightarrow \begin{pmatrix} \begin{pmatrix} 0 & 7 & 42 \\ 1 & 8 & 43 \\ 4 & 11 & 46 \\ 5 & 12 & 47 \end{pmatrix}_{0,0} & \begin{pmatrix} 14 & 21 \\ 15 & 22 \\ 18 & 25 \\ 19 & 26 \end{pmatrix}_{0,1} & \begin{pmatrix} 28 & 35 \\ 29 & 36 \\ 32 & 39 \\ 33 & 40 \end{pmatrix}_{0,2} \\ \begin{pmatrix} 2 & 9 & 44 \\ 3 & 10 & 45 \\ 6 & 13 & 48 \end{pmatrix}_{1,0} & \begin{pmatrix} 16 & 23 \\ 17 & 24 \\ 20 & 27 \end{pmatrix}_{1,1} & \begin{pmatrix} 30 & 37 \\ 31 & 38 \\ 34 & 41 \end{pmatrix}_{1,2} \end{pmatrix}$$

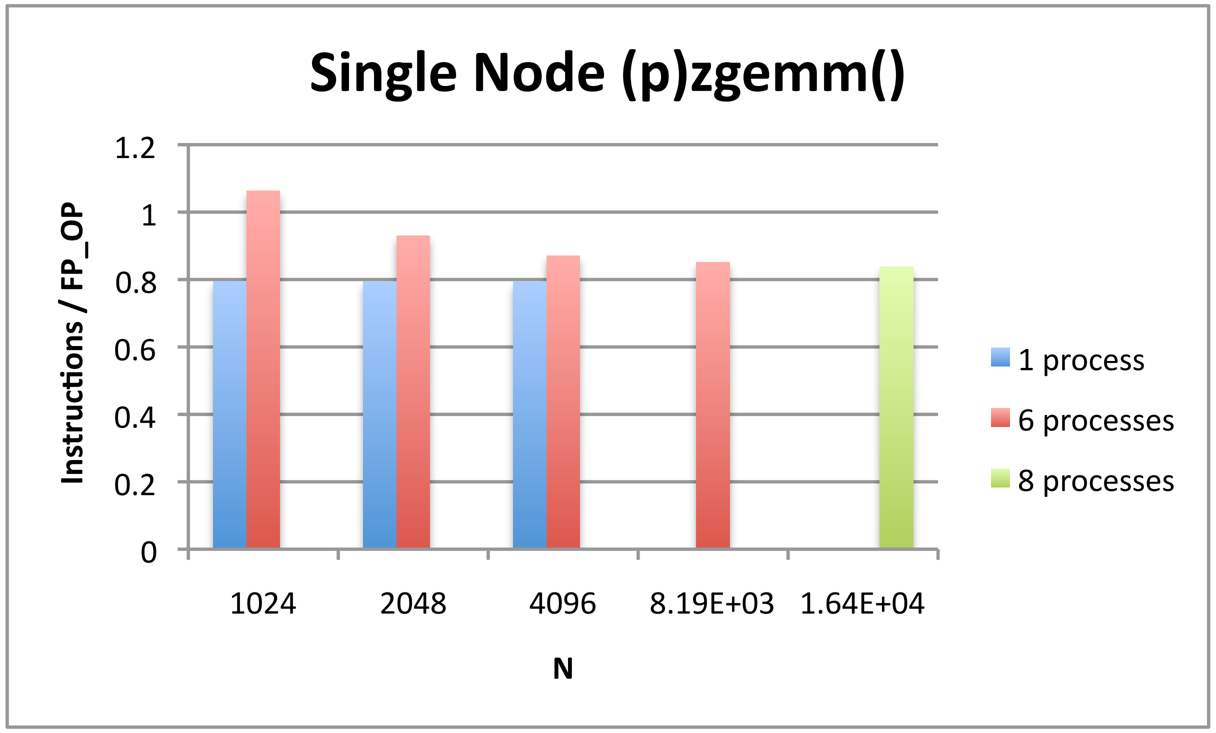
(m,n) = (7,7)
(p,q) = (2,3)
(mb,nb) = (2,2)

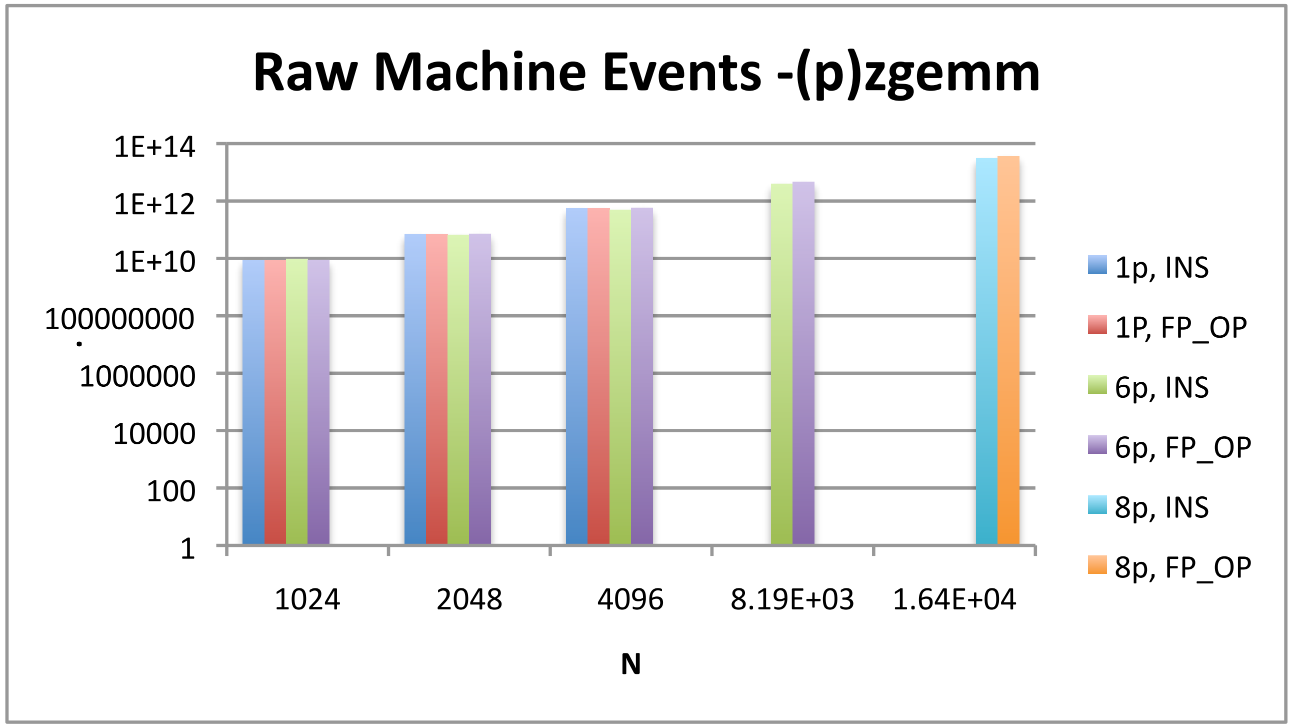
$$\begin{pmatrix} 2 & 9 & 44 \\ 3 & 10 & 45 \\ 6 & 13 & 48 \end{pmatrix}_{1,0} \quad f^{-1}(2d\_block\_cyclic) \longrightarrow A = \begin{pmatrix} * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ 2 & 9 & * & * & * & * & 44 \\ 3 & 10 & * & * & * & * & 45 \\ * & * & * & * & * & * & * \\ * & * & * & * & * & * & * \\ 6 & 13 & * & * & * & * & 48 \end{pmatrix}$$

What We Observe in DOE Apps -that they are Not Usually Dominated by FLOPs

Application	1	2	3	4	5	6	7	8
Instructions Retired	1.99E+15	8.69E+17	1.86E+19	2.45E+18	1.24E+16	7.26E+16	8.29E+18	2.67E+18
Floating Point Ops	3.52E+11	1.27E+15	1.95E+18	2.28E+18	6.16E+15	4.15E+15	3.27E+17	1.44E+18
INS / FP_OP	5.64E+03	6.84E+02	9.56	1.08	2.02	17.5	25.3	1.85

REFERENCE FLOATING POINT INTENSE PROBLEM :: Dense Matrix Matrix Multiplication
C <--- a A B + b C :: OPERATIONAL COMPLEXITY :A[m,n] , B[n,p] , C[m,p] :: [8mpn + 13mp] FLOP
E.g. m=n=p=1024 ---> 8603566080 FLOP , measure 8639217664

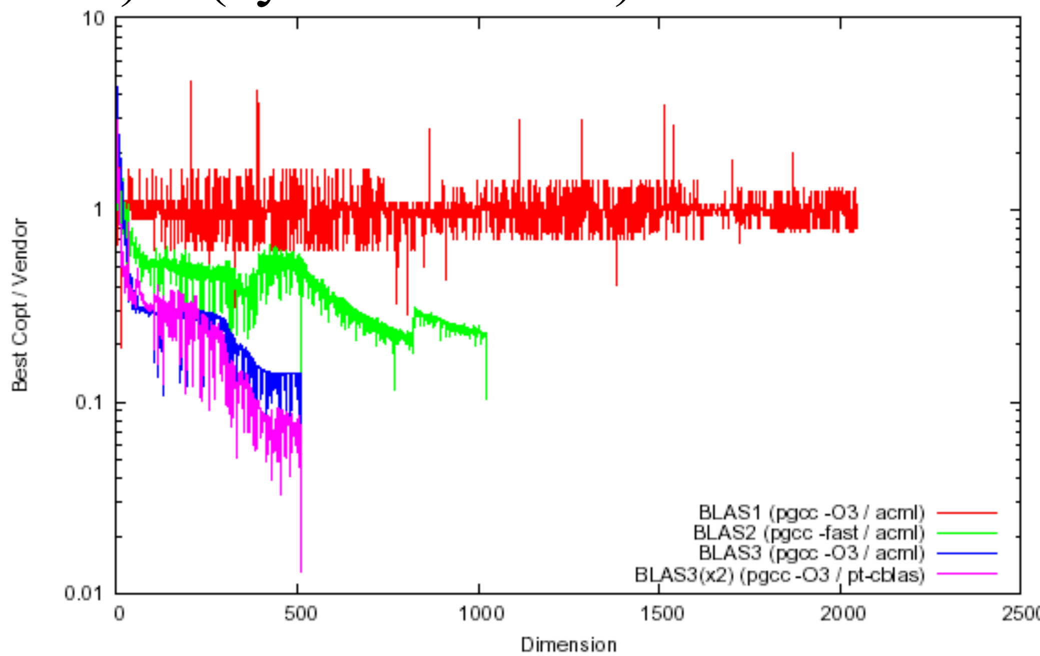




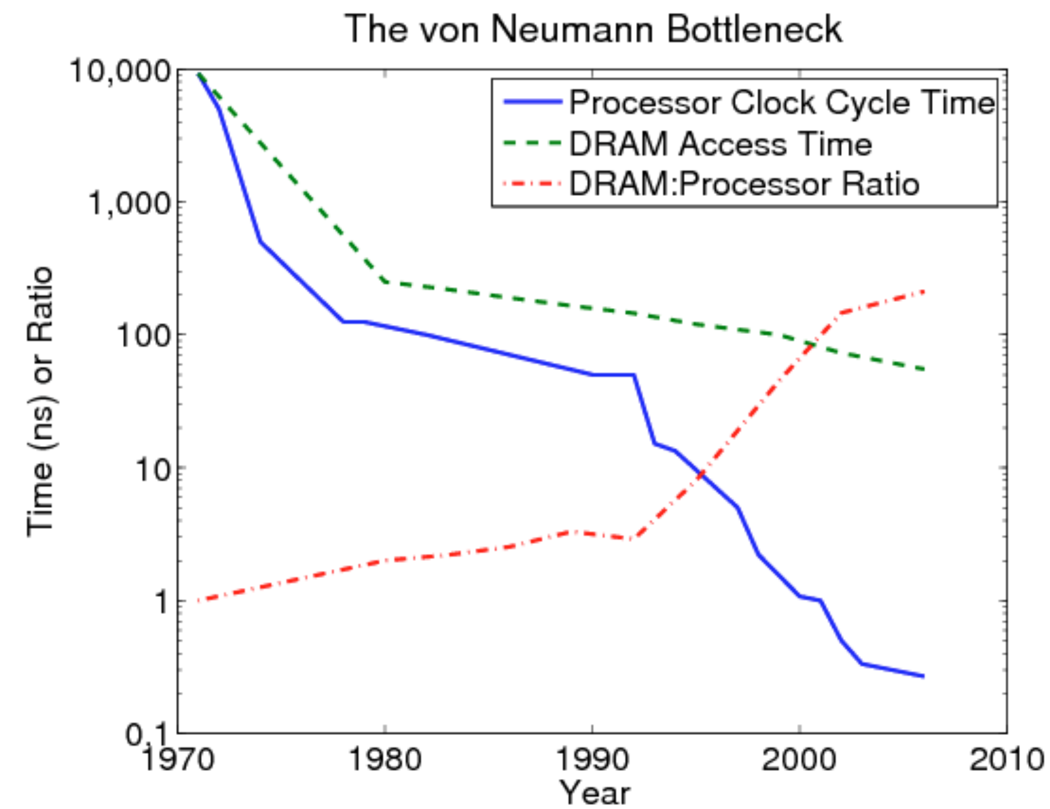
Memory Wall *Always* There ...

Computation: Theoretical peak: (# cpu cores) * (flops / cycle / core) * (cycles / second)

Memory: Theoretical peak: (bus width) * (bus speed)



BLAS 1:	O(n) operations on O(n) operands
BLAS 2:	O(n**2) operations on O(n**2) operands
BLAS 3:	O(n**3) operations on O(n**2) operands



y = α *x* + *y* :

3 loads, 1 store

(more expensive than FP_OPs by a long shot)

2 floating point operations (maybe 1) on 3 operands

eg, double precision on the FY10 target platform:

(3 operands / 2 flop) * (8 bytes / operand) * 6 core * 4 (flop / cyc /core) * 2.6e9(cyc/sec) ~125 GBps

... We don't have this and to get it is $$$ ... how to achieve **Sustainability??**

Basic Optimizations (repeated themes: ***concurrency, atomicity, and bandwidth***)

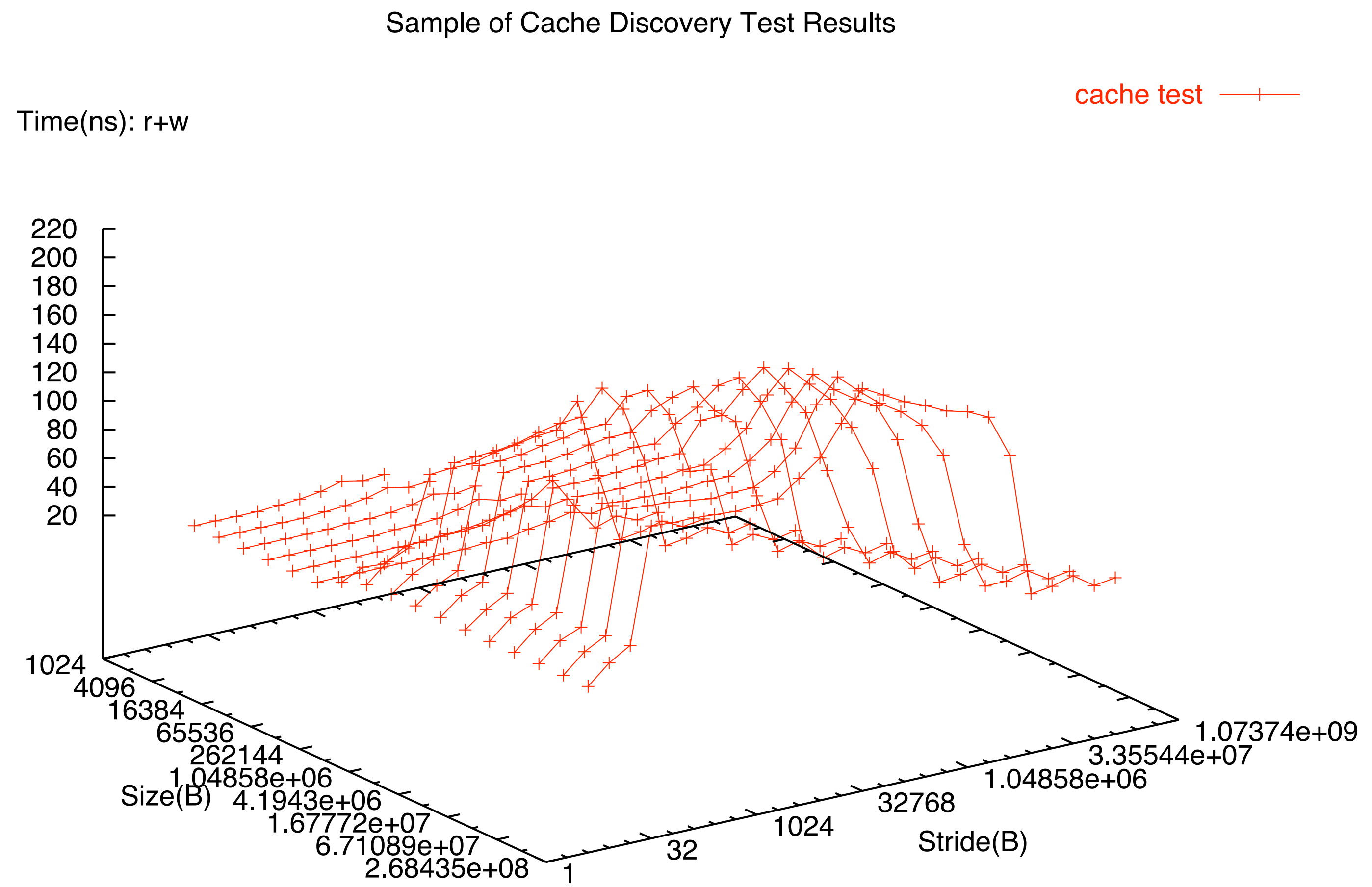
- build a picture of how ***threads*** use memory
  - -locality, latency, bandwidth, coherency, cache contention

- understand program, execution / use of programming model
  - -delay error norms in iterative convergence, precompute interpolation / derivative coefficients, discretization representations (ie improved unit cells, exploiting spatial homogeneties)

  - -concurrency, balanced distributed parallelism, communication (blocking send receive pairs, barrier removal, collectives), control flow dependencies (mutex, semaphore, synchronizations) and i/o issues

temporal locality
when a referenced resource is referenced again sometime in the near future

spatial locality
the chance of referencing a resource is higher if a resource near it was just referenced



Cache Coherency:

write-through, if cache line is written to, the processor also writes to main memory (at all times cache and memory are in synche)

write-back, cache line is marked dirty, write back is delayed to when cache line is being evicted

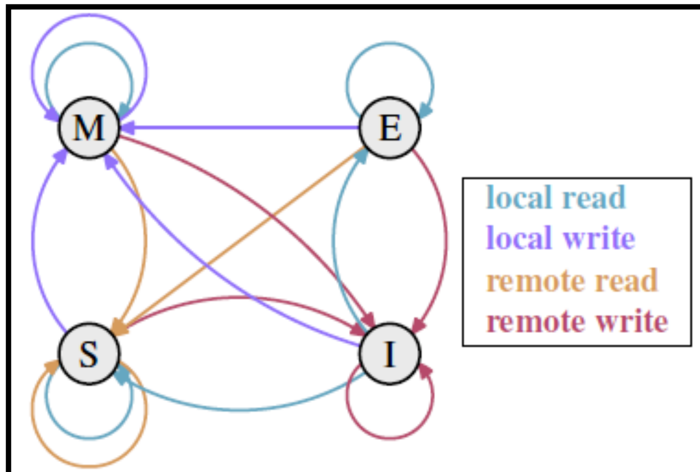
> >1 processor core is active (say in SMP) -all processors still have to see the same memory content; have to exchange CL when needed -includes the MC

write-combining (ie on graphics cards)

set-associative dereferencing (the larger the set and CL, the fewer the misses):
tag and data in sets -a set maps to the address of the cache line, a small number of values is cached for the same set value ; the tags for all such sets are compared in parallel
ie 8 sets for L1 and 24 associativity levels for L2 are common;
for 4MB/64B and 8 way set-associativity then 8192 sets (requires 13bit address tag) ; to find if the address is in cache only 8 tags have to be compared!

Use of threads means coping with complicated issues

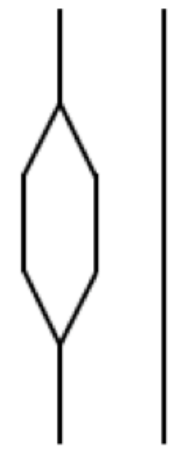
- cache contention, coherency
- memory bandwidth
- scheduling



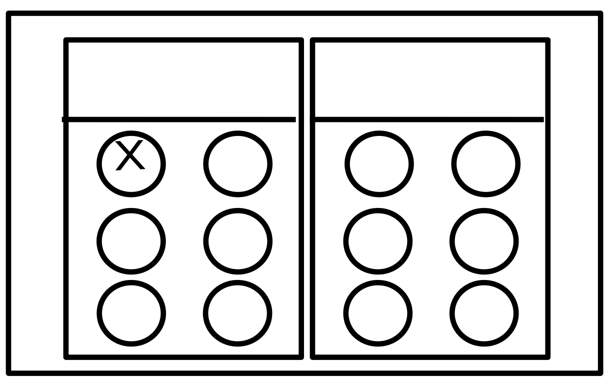
Modified, local processor has only copy of data and modifies it
Exclusive, CL is not modified and not in another core's cache
Shared, CL not modified -might be in cache somewhere
Invalid, CL is invalid -not used

*other processor's activities are snooped on the address bus

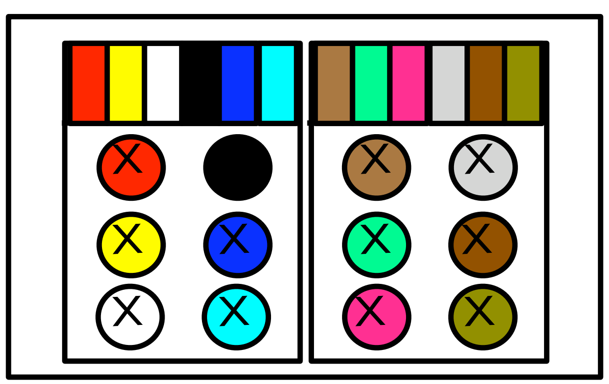
fork (create) / join overheads



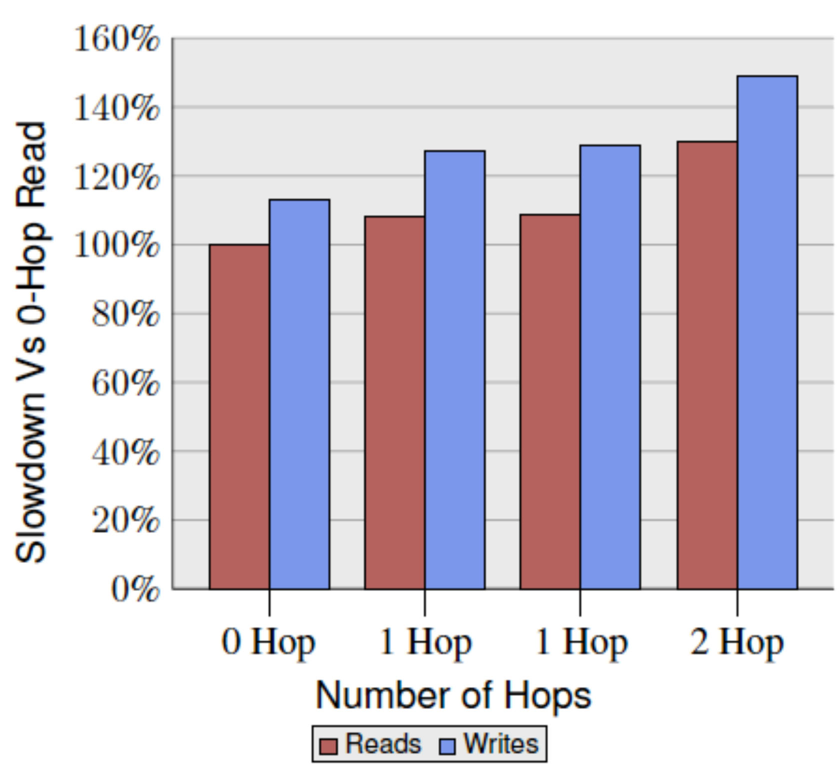
NT	Cycles	L2DCM
1	1959379	69
2	2020818	81
4	2289393	122
6	2366367	146
8	2499159	239



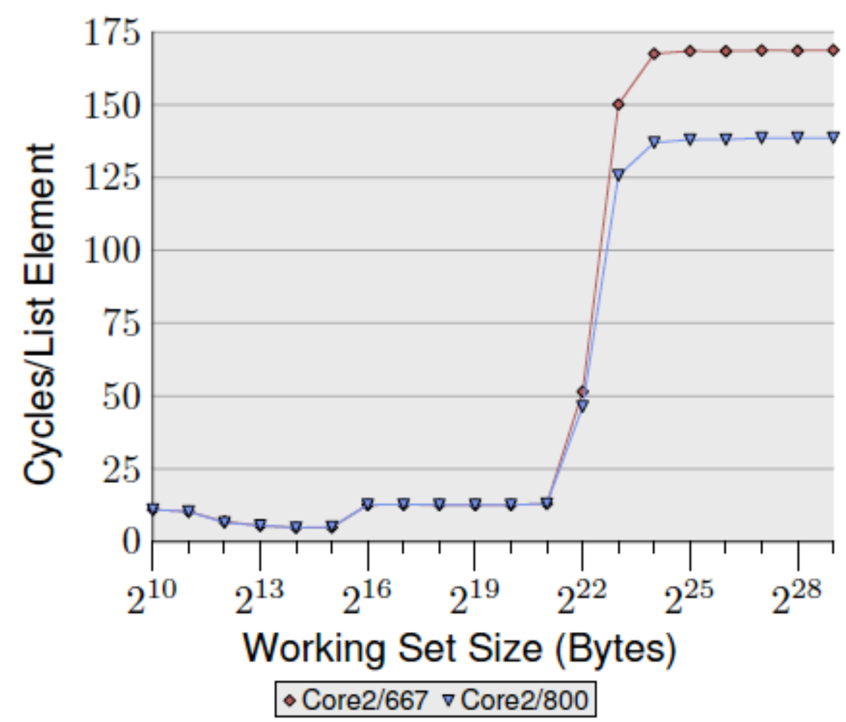
NUMA + memory affinity



no NUMA, 6 PEs/socket



make the FSB faster with increasing core count



Basic Optimizations (repeated themes: *concurrency, atomicity,* and *bandwidth*)

base / node focus

- **non-temporal writes,** ie don't cache the data writes since it won't be used again soon (i.e. n-tuple initialization)
  - avoids reading cache line before write, avoids wasteful occupation of cache line and time for write (memset()); does not evict useful data
  - sfence() compiler set barriers
- **loop unrolling** , transposing matrices
- **vectorization**, 2,4,8 elements computed at the same time (SIMD) w/ multi-media extensions to ISA
- **reordering** elements so that elements that are used together are stored together -pack CL gaps w/ usable data (i.e. try to access structure elements in the order they are defined in the structure)
- **stack alignment,** as the compiler generates code it actively aligns the stack inserting gaps where needed ... is not necessarily optimal -if statically defined arrays, there are tools that can improve the alignment; separating n-tuples may increase code complexity but improve performance
- **function inlining**, may enable compiler or hand -tuned instruction pipeline optimization (ie dead code elimination or value range propagation) ; especially true if a function is called only once
- **prefetching**, hardware, tries to predict cache misses -with 4K page sizes this is a hard problem and costly penalty if not well predicted; software (void _mm_prefetch(void *p, enum _mm_hint h) --_MM_HINT_NTA -when data is evicted from L1d -don't write it to higher levels)

Loop fusion transforms multiple distinct loops into a single loop. It increases the granule size of parallel loops and exposes opportunities to reuse variables from local storage. Its dual, loop distribution, separates independent statements in a loop nest into multiple loops with the same headers.

```
PARALLEL DO I = 1, N
      A(I) = 0.0                          PARALLEL DO I = 1, N
END                    ⟹                       A(I) = 0.0
                     fusion                     B(I) = A(I)
PARALLEL DO I = 1, N                      END
      B(I) = A(I)      ⟸
END                 distribution
```

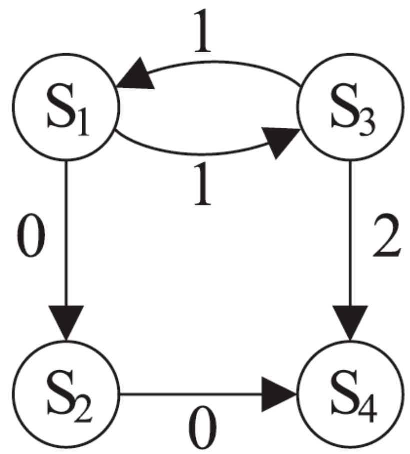
In the example above, the fused version on the right experiences half the loop overhead and synchronization cost as the original version on the left. If all A(1:N) references do not fit in cache at once, the fused version at least provides reuse in cache. Because the accesses to A(I) now occur on the same loop iteration rather than N iterations apart, they could also be reused in a register. For sequential ex-

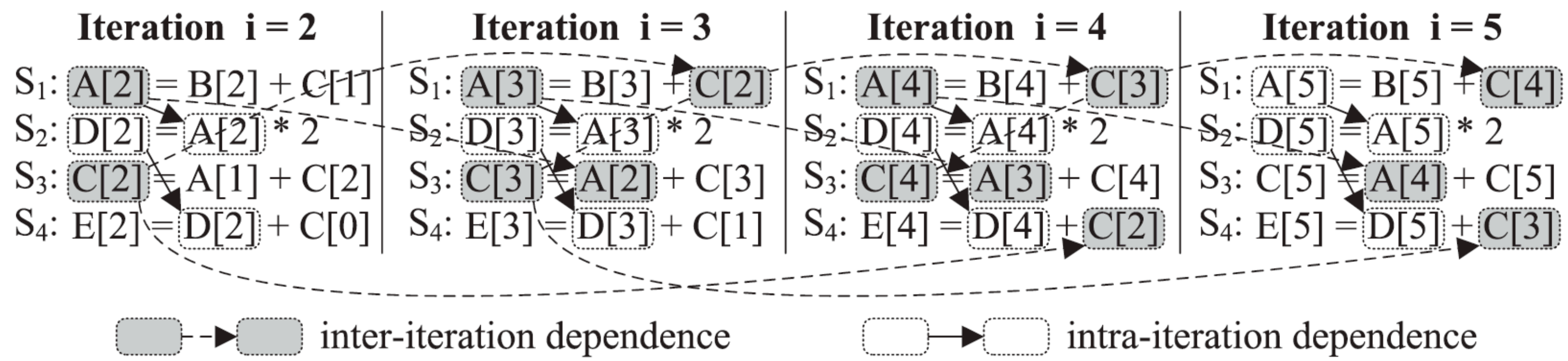
source: K. Kennedy, *Rice*

- reduce synchronization overheads in parallel loops
- improve data locality

Going Beyond Instruction Level //ism to Loop Level

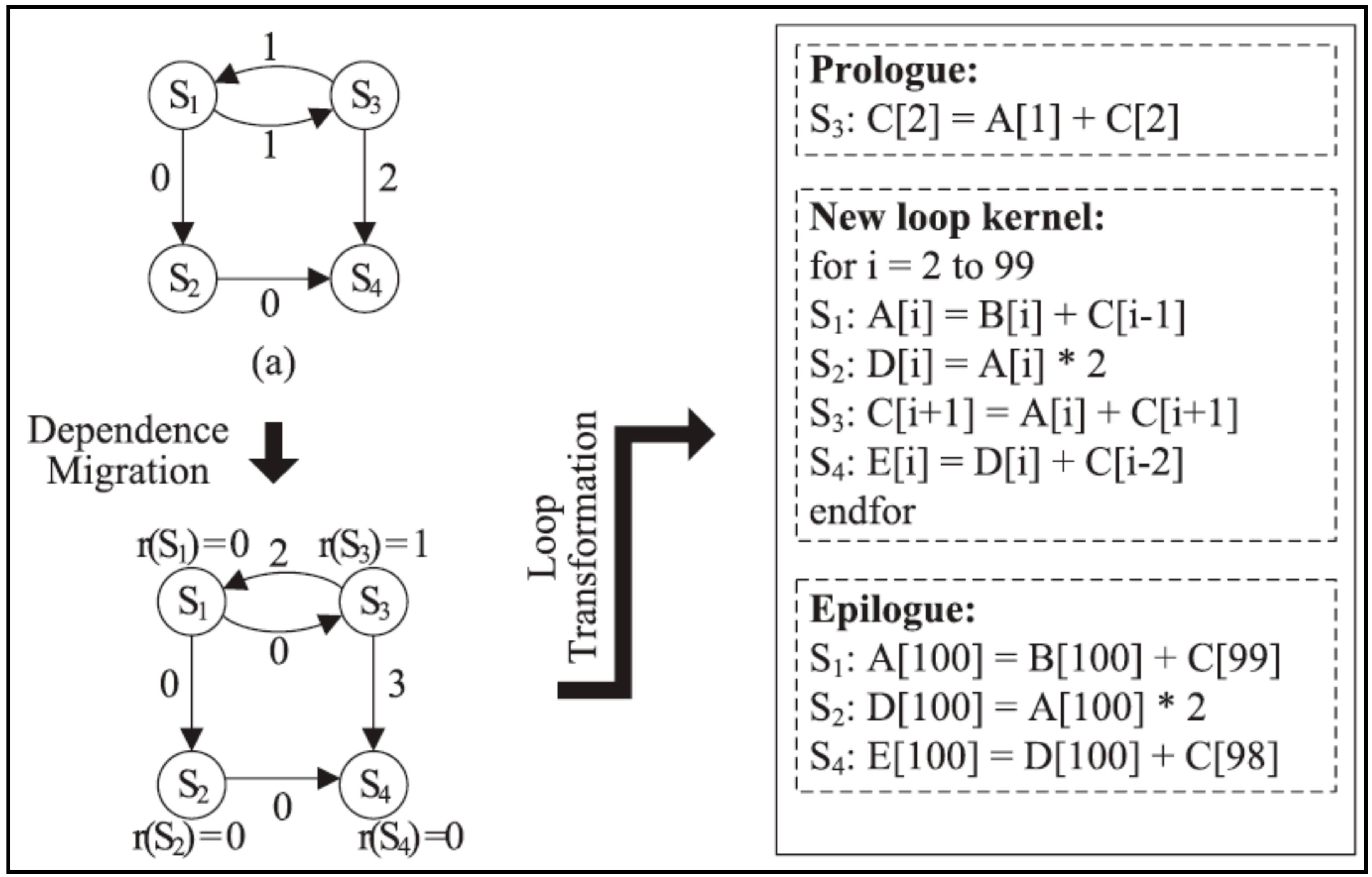
```
for i = 2 to 100
S1: A[i] = B[i] + C[i-1]
S2: D[i] = A[i] * 2
S3: C[i] = A[i-1] + C[i]
S4: E[i] = D[i] + C[i-2]
endfor
```





before, minimum nonzero edge weight = 1

after, minimum nonzero edge weight = 2



Optimally Maximizing Iteration-Level Loop Parallelism, D. Liu et al., IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 23, NO. 3, MARCH 2012

Unblock Communications if Possible

Blocking

```
if ( ip % 2 )
  { /* BLOCKING */
  MPI_Send( sbf , n , MPI_DOUBLE , ngh[ 0 ] , itag , MPI_COMM_WORLD ) ; /* send to left */
  MPI_Recv( rbf , n , MPI_DOUBLE , ngh[ 1 ] , itag , MPI_COMM_WORLD , &mpi_st ) ; /* receive from right */
  MPI_Send( sbf + n , n , MPI_DOUBLE , ngh[ 1 ] , itag , MPI_COMM_WORLD ) ; /* send to right */
  MPI_Recv( rbf + n , n , MPI_DOUBLE , ngh[ 0 ] , itag , MPI_COMM_WORLD , &mpi_st ) ; /* receive from left */
  }
else
  {
  MPI_Recv( rbf , n , MPI_DOUBLE , ngh[ 1 ] , itag , MPI_COMM_WORLD , &mpi_st ) ; /* receive from right */
  MPI_Send( sbf , n , MPI_DOUBLE , ngh[ 0 ] , itag , MPI_COMM_WORLD ) ; /* send to left */
  MPI_Recv( rbf + n , n , MPI_DOUBLE , ngh[ 0 ] , itag , MPI_COMM_WORLD , &mpi_st ) ; /* receive from left */
  MPI_Send( sbf + n , n , MPI_DOUBLE , ngh[ 1 ] , itag , MPI_COMM_WORLD ) ; /* send to right */
  }
```

Non-Blocking

```
  { /* ASYNCHRONOUS  */

    MPI_Isend( sbf , n , MPI_DOUBLE , ngh[ 0 ] , itag , MPI_COMM_WORLD , r ) ; /* send to the left */
    MPI_Isend( sbf + n , n , MPI_DOUBLE , ngh[ 1 ] , itag , MPI_COMM_WORLD , r + 1 ) ; /* send to the right */
    MPI_Irecv( rbf , n , MPI_DOUBLE, ngh[ 1 ] , itag , MPI_COMM_WORLD , r + 2 ) ; /* receive from the right */
    MPI_Irecv( rbf + n , n , MPI_DOUBLE , ngh[ 0 ] , itag , MPI_COMM_WORLD , r + 3 ) ; /* receive from the left */
    MPI_Waitall( 4 , r , _st ) ;
  }
```

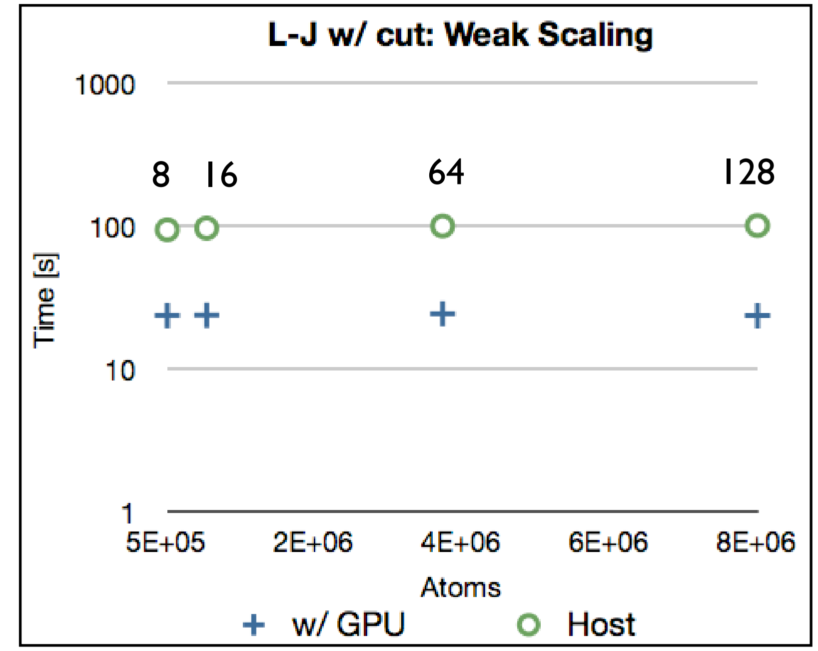
nn exchanges > 2X performance gain, same results!

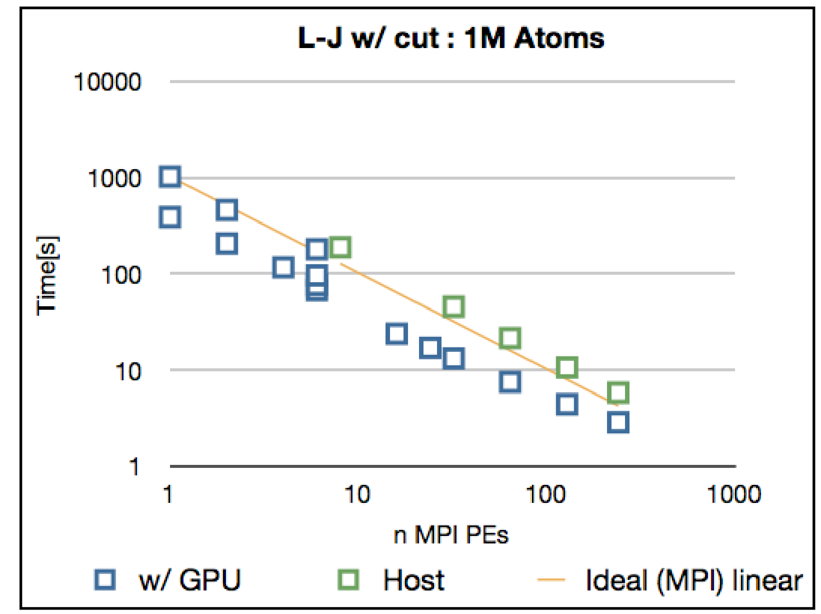
Exploit Multi-core Hybrid Programming Model

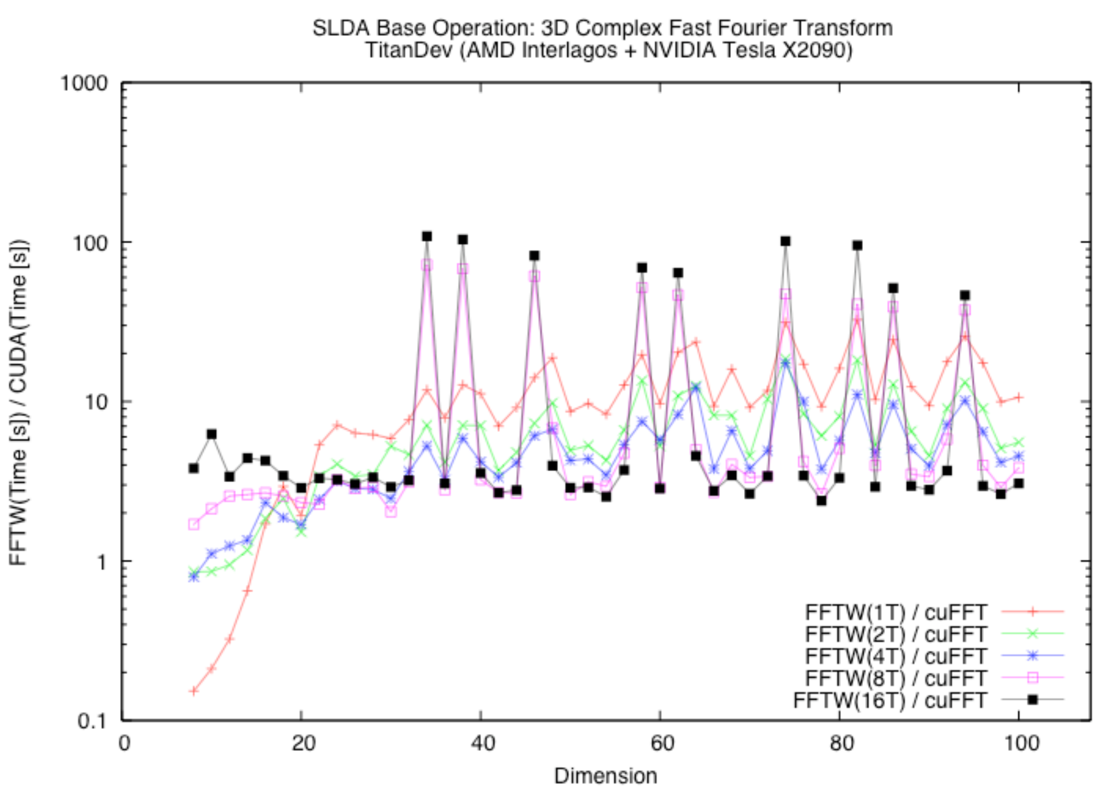
- **MPI processes** spawn lightweight processes
- **OpenMP threads**, #include <omp.h> , omp_set_num_threads();
- **POSIX threads**, #include <pthread.h> , pthread_create();
- **CUDA,** kernel execution

-lsize=16	MPI	LWP	DRAM
aprun -n <1-16>	1 - 16	1	2 * 2^30
aprun -n 2 -sn 2 -S 1 -d 8	2	1 - 8	16 * 2^30
aprun -n 1 -N 1 -d 16	1	1 - 16	32 * 2^30

<-S> * <-d> cannot exceed the maximum number of CPUs per NUMA node

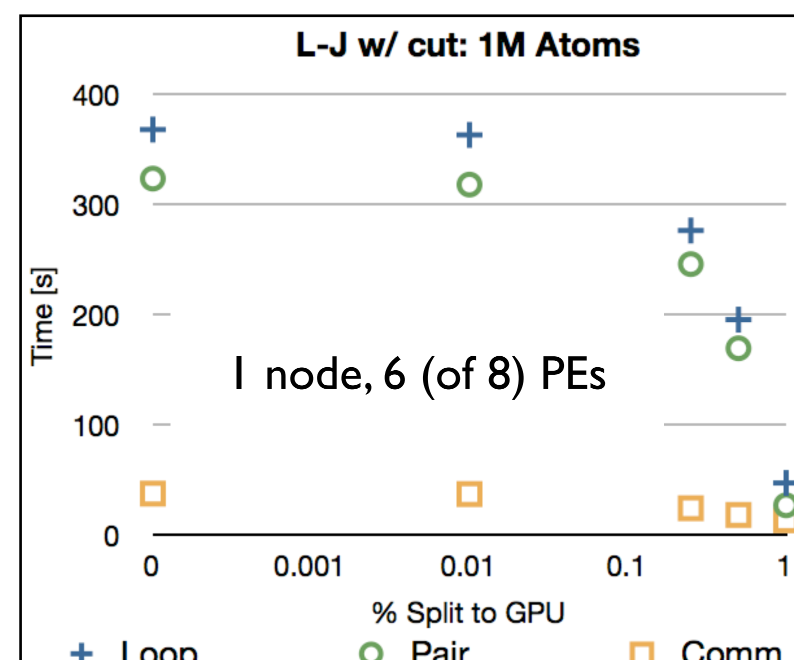






$$4\epsilon[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}]$$

Lennard-Jones (12,6)



OSIRIS: Laser Wakefields (detailed example from FY11 DOE ASCR OMB software metric study)

How does a short and intense driver evolve over large distances?
How is the wake excited and how does it evolve?
How do the properties of the witness beams evolve as they are accelerated?

• short and intense laser or relativistic particle beams propagate through a plasma near the speed of light

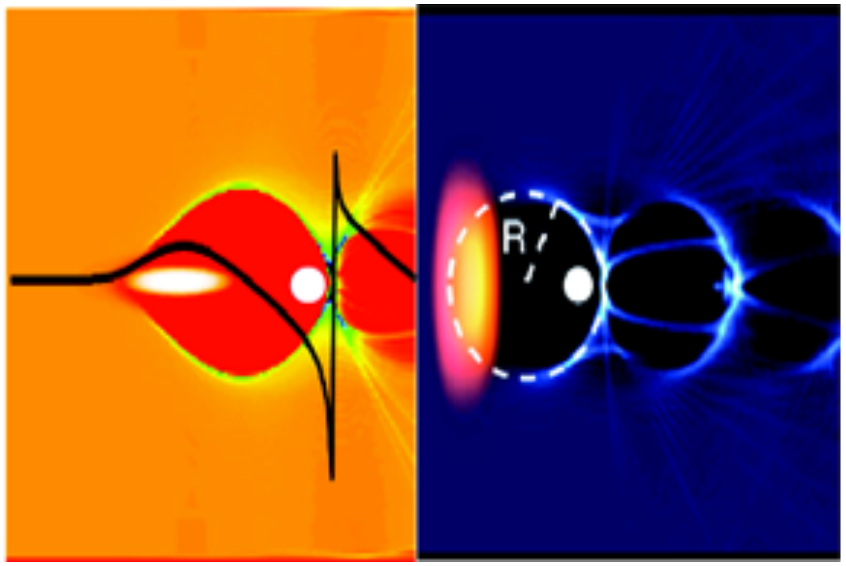
• light pressure of the laser or the space charge forces from the particle beam displaces plasma electrons

• the ions pull the electrons back towards where they started creating a plasma wave wake with a phase velocity near the speed of light

• accelerating (electric) fields in these wakes are more than 1000 times higher than those in existing accelerators.

• properly shaped and phased electrons or positron beams (witness beams) are loaded onto the wake and they surf to ultra-high energies in very short distances.

• Experiments using a laser driver have demonstrated the feasibility of generating GeV class quasi-monoenergetic beams

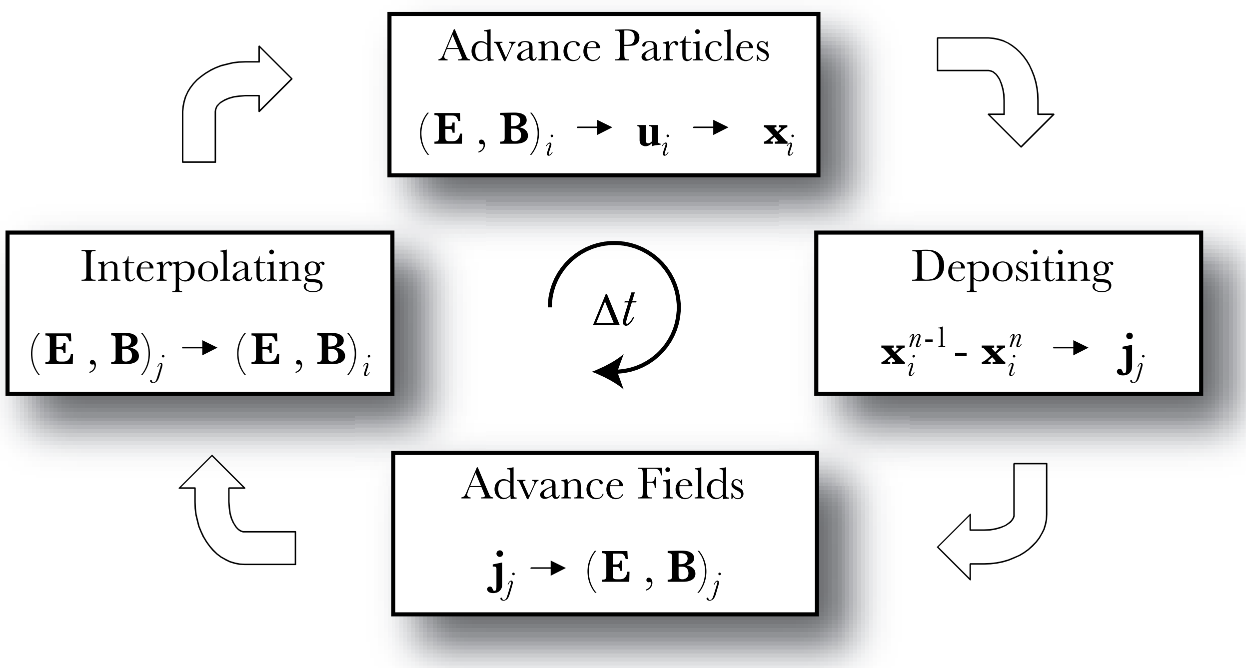


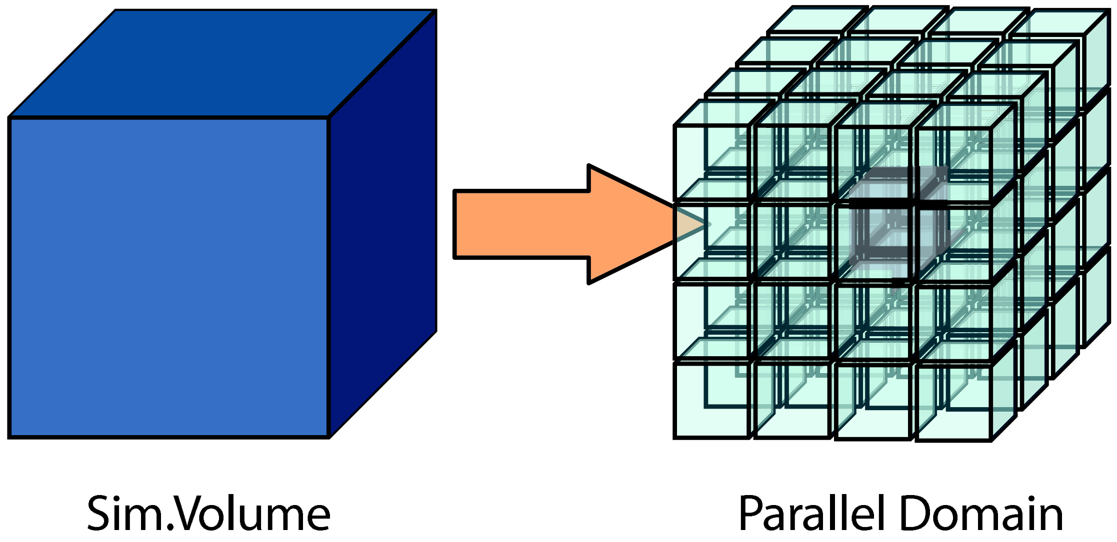
On the left is an electron beam (white) moving from right to left.
It forms a wakefield (density of plasma is shown. A lineout of the accelerating field is shown in black. A trailing bunch is shown in white in the back of the wakefield. On the right a laser (orange) is moving from right to left. It also creates a wakefield. The wakefield in both cases is a moving bubble of a radius R. A trailing beam is shown in white as well.

OSIRIS:

The fields within the wake structure demand a full electromagnetic treatment is needed.

The leading kinetic description is the particle-in-cell (PIC) method.

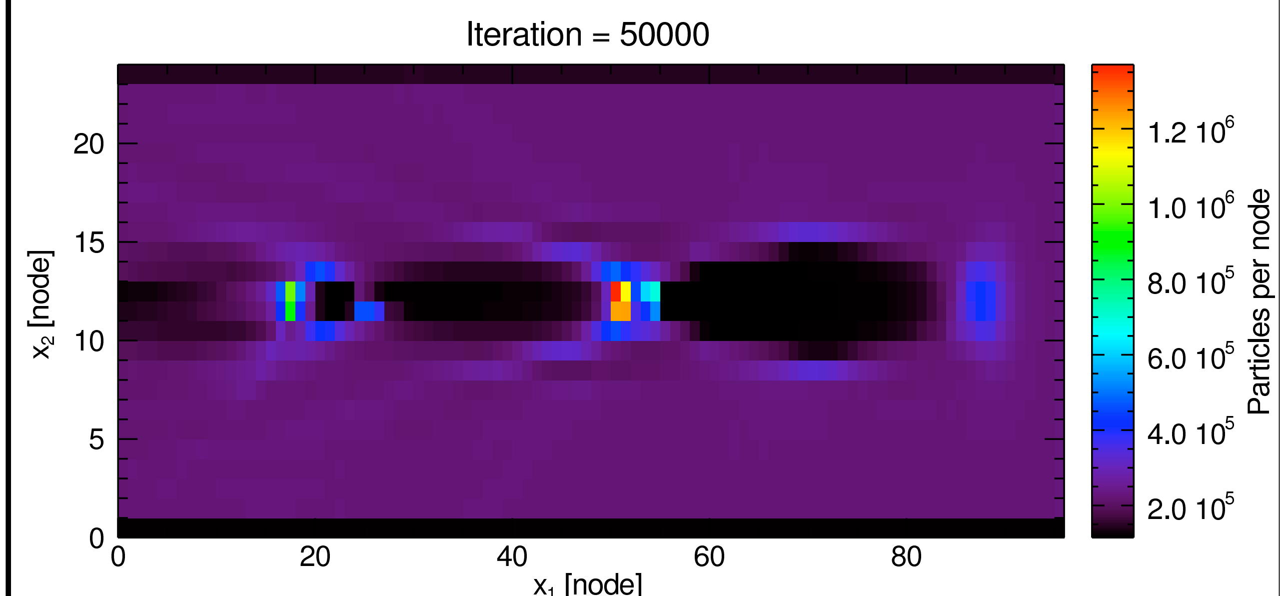




•deposit some particle quantity, such as a charge, is accumulated on a grid via interpolation to produce a source density. Various other quantities can also be deposited, such as current densities

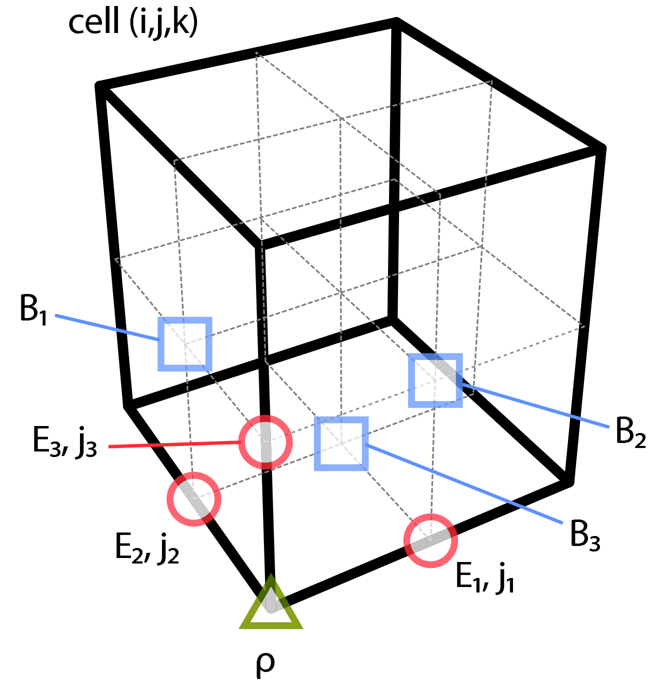
•field solver, which solves Maxwells equations or a subset to obtain the electric and/or magnetic fields from the source densities

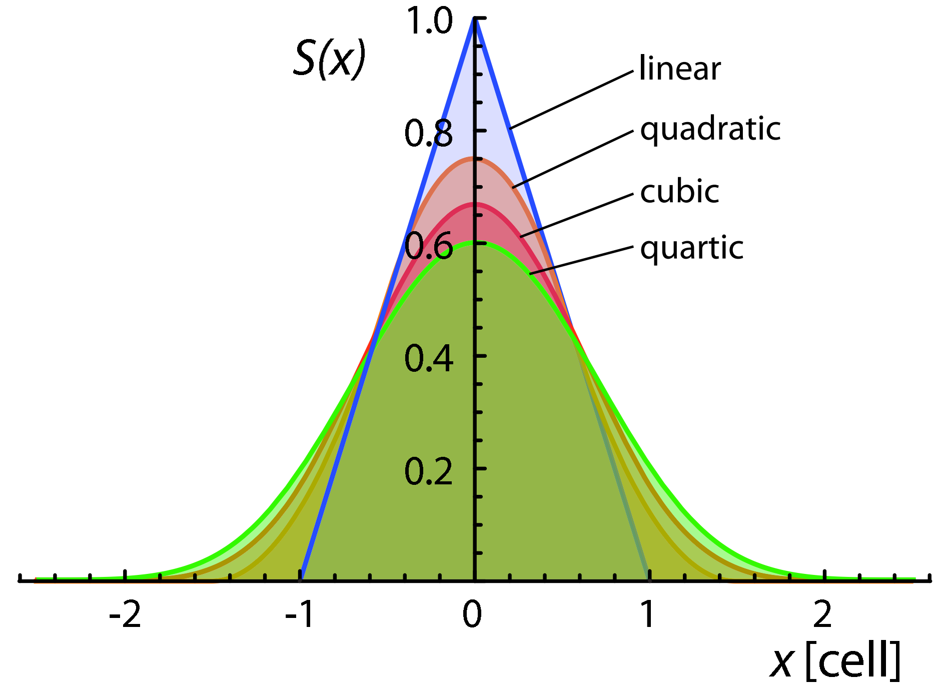
•particle forces are found by interpolation from the grid, and the particle coordinates are updated, using Newtons second law and the Lorentz force. The particle processing parts dominate over the field solving parts



Balancing the particle load is hard problem!

OSIRIS:





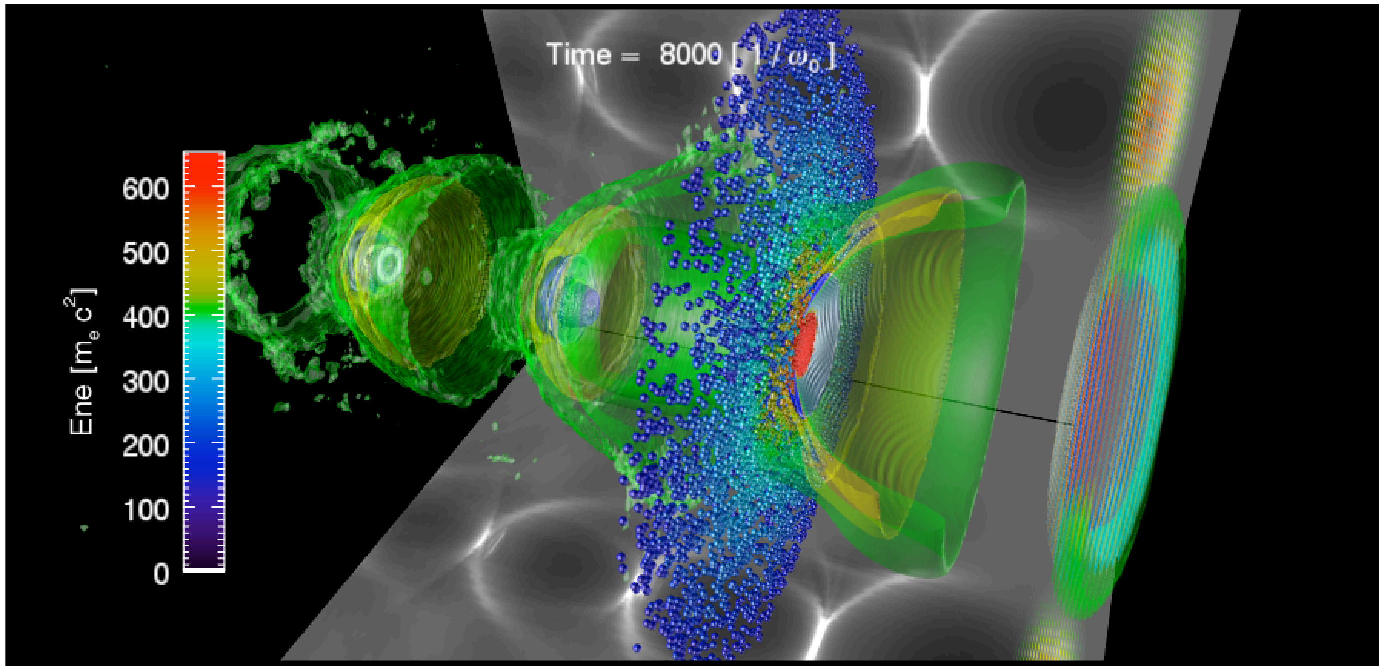
	linear	quadratic	cubic	quartic
S_{-2}				$\frac{1}{384}(1-2x)^4$
S_{-1}		$\frac{1}{8}(1-2x)^2$	$\frac{1}{6}(1-x)^3$	$\frac{1}{96}\left(-16x^4+16x^3+24x^2-44x+19\right)$
S_0	$1-x$	$\frac{3}{4}-x^2$	$\frac{1}{6}\left(3x^3-6x^2+4\right)$	$\frac{1}{4}x^4-\frac{5}{8}x^2+\frac{115}{192}$
S_1	x	$\frac{1}{8}(1+2x)^2$	$\frac{1}{6}\left(-3x^3+3x^2+3x+1\right)$	$\frac{1}{96}\left(-16x^4-16x^3+24x^2+44x+19\right)$
S_2			$\frac{1}{6}x^3$	$\frac{1}{384}(1+2x)^4$

- need a method to effectively connect grid and particles quantities to determine the force acting on the particle.

- field interpolation calculations require knowledge of the grid point index closest to the particle position, and the distance between the particle and the grid point, normalized to the cell size.

- OSIRIS implements 1st to 4th order interpolation schemes (linear, quadratic, cubic and quartic splines)

OSIRIS: Problems



Uniform Plasma

•**(1)** warm plasma with a temperature distribution parameter of u_thermal = 0.01c

• a perfectly load balanced simulation

• particle diffusion across parallel nodes happens uniformly so the total number of particles per node remains approximately constant.

• good performance test as these plasma conditions

• resemble those on most of the simulation box for the laser wakefield runs.

*quadratic shaped particles for the current deposition and field interpolation for all the simulations

Laser Wakefield scenarios

•**(2,3)** interaction of a 200 TW (6 Joule) laser interacting with uniform plasma with a density of 1.5e18 cm^-3

•plasma with an intensity sufficient to trigger self-injection, under different numerical and physical conditions.

•different grid resolutions, different number of particles per cell, and mobile/immobile ions.

•**(4)** a PW (30J) laser propagating in a .5e 18 cm^-3 plasma where ion motion is expected to play an important role

Run	Grid	Simulation Box $[c/\omega_0]$	Particles	Iterations	Laser a_0	Ions
Warm test	$6144 \times 6144 \times 1536$	$614.4 \times 614.4 \times 153.6$	4.46×10^{11}	5600	n/a	n/a
Run 1	$8064 \times 480 \times 480$	$806.4 \times 1171.88 \times 1171.88$	3.72×10^{9}	41000	4.0	fixed
Run 2	$8832 \times 432 \times 432$	$1766.4 \times 2041.31 \times 2041.31$	6.59×10^{9}	47000	4.58	fixed
Run 3	$4032 \times 312 \times 312$	$806.4 \times 1171.88 \times 1171.88$	1.26×10^{10}	52000	4.0	moving

OSIRIS: Enhancements

SIMD Optimizations and SSE Implementation

• 90 / 10 rule - advancing particles and deposting the current

• optimized the use of memory and L2 cache for vector version

• store individual components in separate sequential arrays
 -one for x, one for y and one for z

particles:
i) load 4 particles into the vector unit
ii) interpolate the EM fields for these 4 particles
iii) push the 4 particles
iv) create up to 4 × 4 virtual particles for current deposition
v) store the 4 particles back to main memory.

virtual particles:
i) load 4 virtual particles into the vector unit
ii) calculate the current contribution for the 4 virtual particles
iii) accumulate this current in the global electric current grid

• make use of vector shuffle operation to efficiently exchange parts of the vector registers:
 i) we read 3 vectors (12 positions) sequentially
 ii) shuffle them to get a vector of 4 x positions,
 one vector of 4 y positions, one vector of 4 z positions

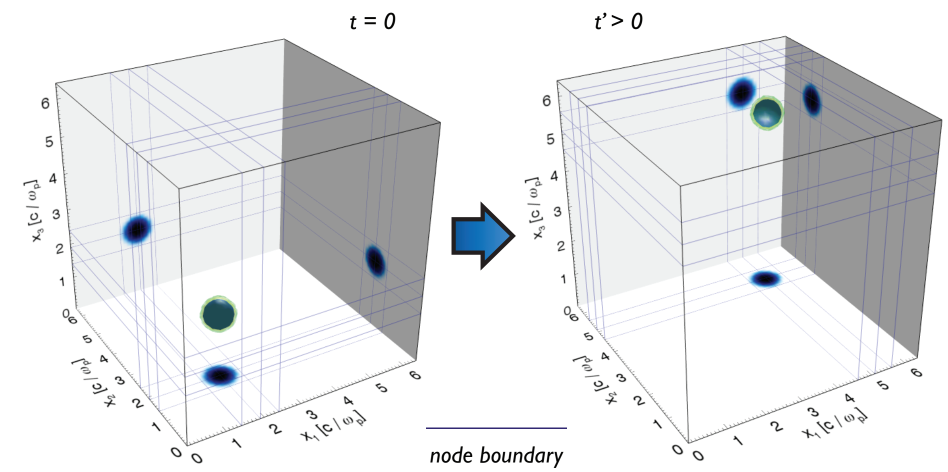
• 4 × 3 transpose is done in the registers and is very efficient (10 cycles overhead)
 -enables efficient use of vector memory read operations

•storing the particles back to memory, the opposite operation is performed

OSIRIS: Other Enhancements

Dynamic Load balancing

- 30% improvement in imbalance, but a 5% drop in overall performance
  - i) determine best partition from current load
  - ii) redistribute boundaries



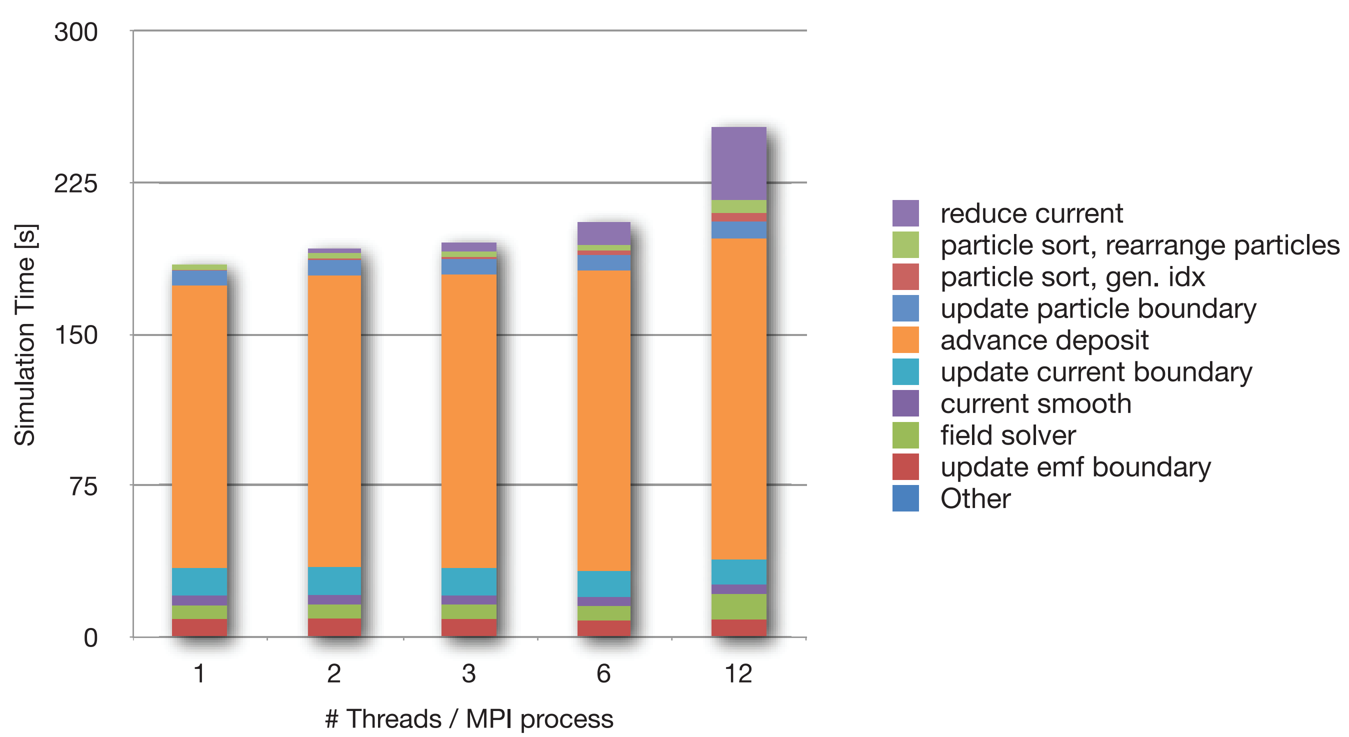
SMP version of major distributed kernels

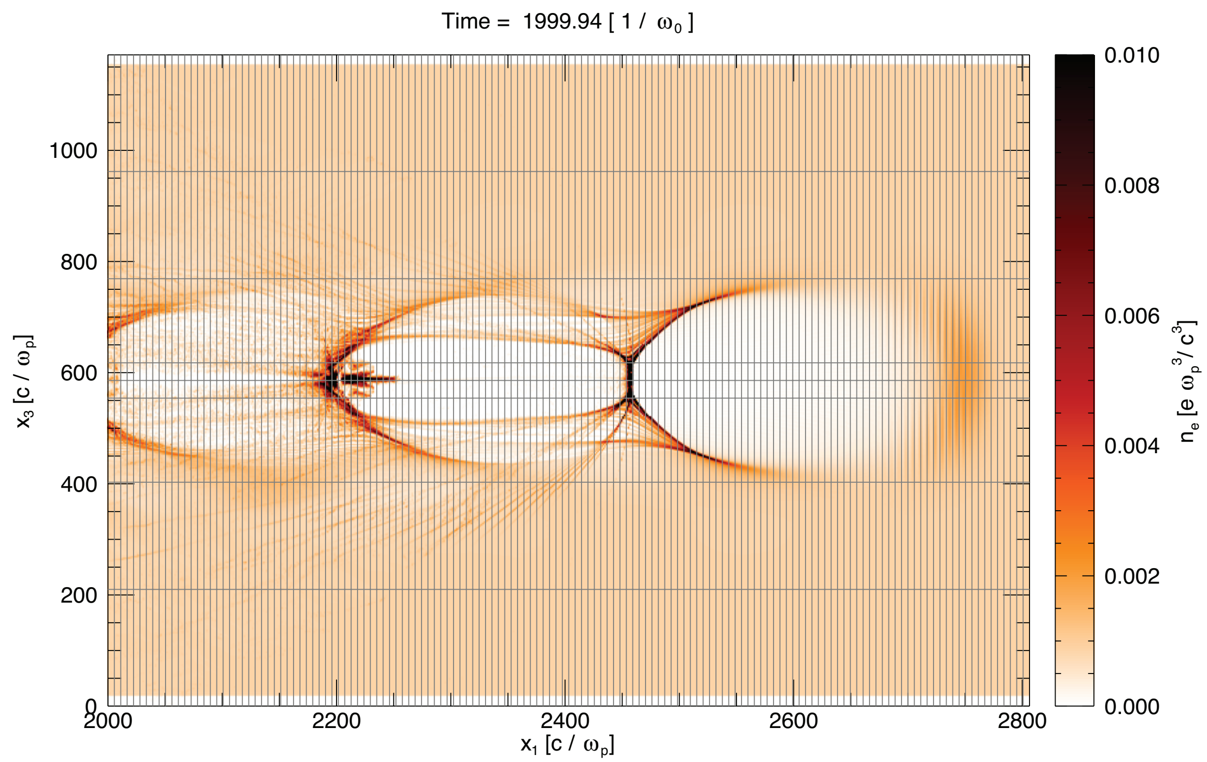
- the volume handled by each group of cores is much larger,
- the probability for significant load imbalance will be lower

- particle pusher, the field solver, current smoother, boundary processing of particles / fields and particle sorting.

- fairly simple since routines generally consist of an external loop that can be easily split among threads

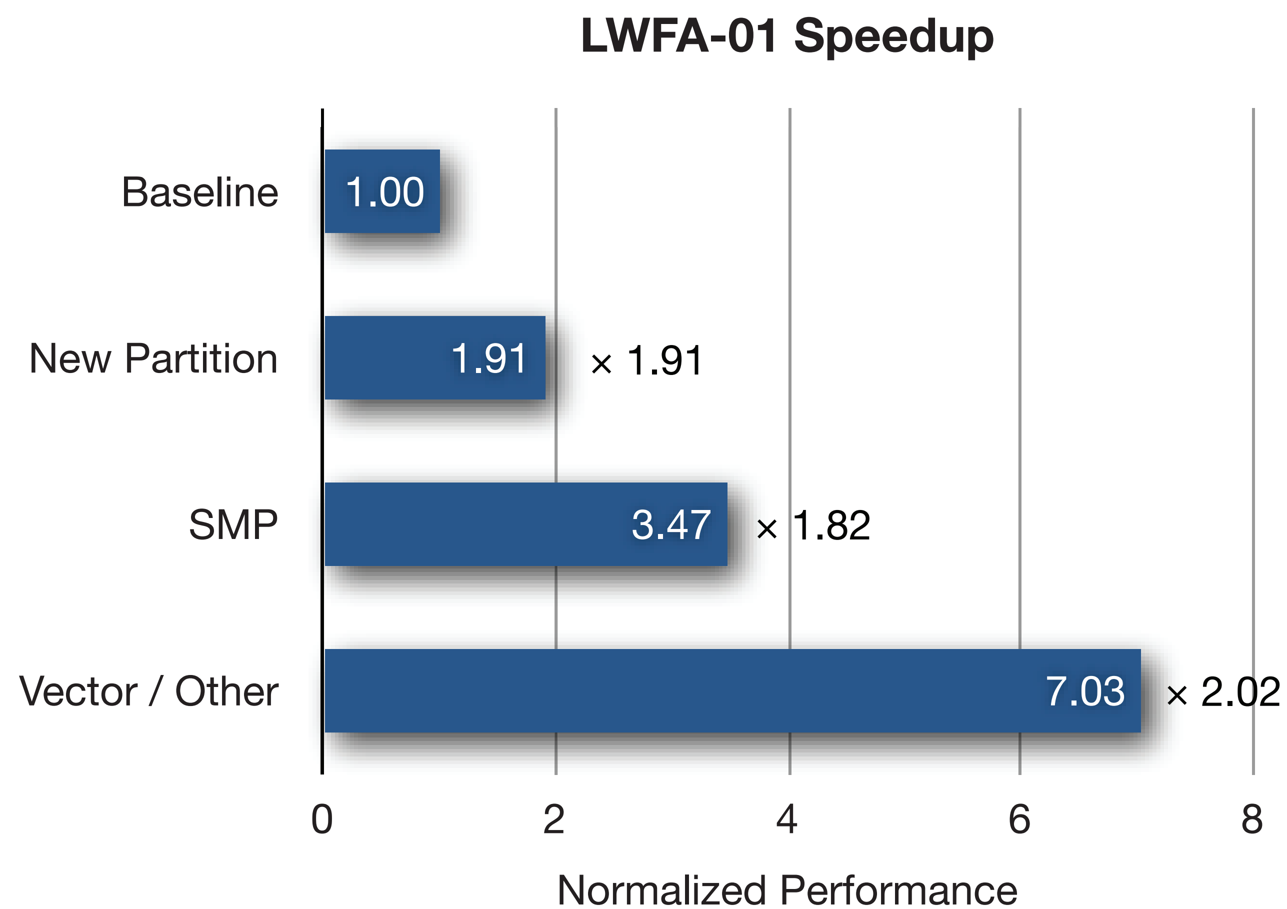
- reduced the total node communication volume
- threads per MPI process must match the number of cores per cpu -or less





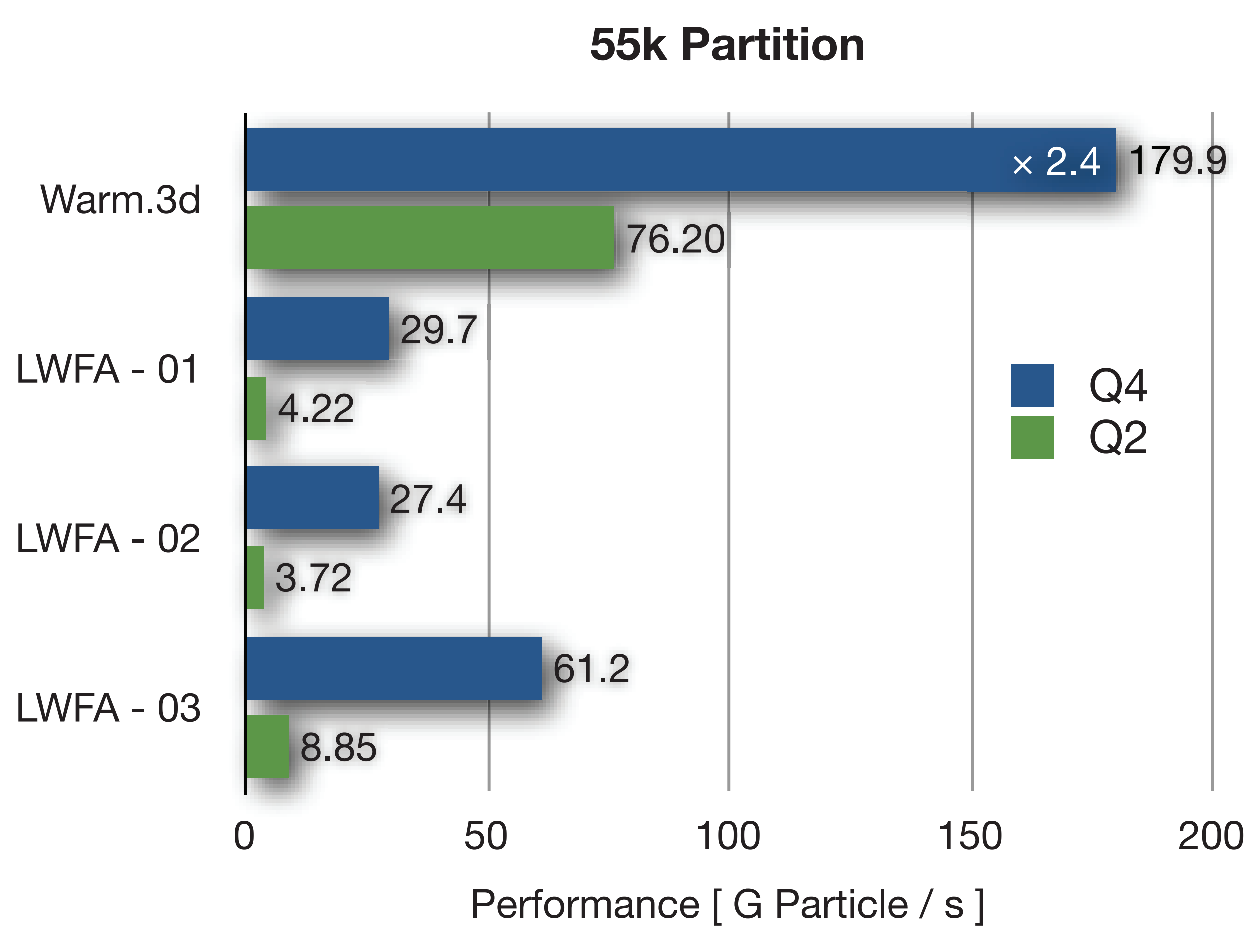
OSIRIS : Particle Injection in Laser Wakefield

Run	Partition [cores]	Performance [G part/s]	Push Time [μs]	Average Imbalance	TFLOPS	INS/FP	Speedup
Warm.3d	55296	179.95	0.307	1.00	169.92	1.28	2.36
LWFA - 01		29.66	1.864	3.64	31.18	6.39	7.03
LWFA - 02		27.43	2.016	4.75	28.02	7.69	7.37
LWFA - 03		61.20	0.903	2.31	58.25	3.84	6.92
Frozen.3d linear	221184	1463.52	0.151	1.00	516.92	1.34	n / a
Frozen.3d quadratic		784.04	0.282	1.00	736.12	1.20	n / a
Warm.3d weak scale		741.20	0.298	1.00	700.09	1.21	9.73
Warm.3d strong scale		719.80	0.307	1.00	679.68	1.28	9.45
LWFA - 01 - strong scale		70.91	3.119	4.66	76.55	9.48	16.80



OSIRIS : Particle Injection in Laser Wakefield

Run	Partition [cores]	Performance [G part/s]	Push Time [μs]	Average Imbalance	TFLOPS	INS/FP	Speedup
Warm.3d	55296	179.95	0.307	1.00	169.92	1.28	2.36
LWFA - 01		29.66	1.864	3.64	31.18	6.39	7.03
LWFA - 02		27.43	2.016	4.75	28.02	7.69	7.37
LWFA - 03		61.20	0.903	2.31	58.25	3.84	6.92
Frozen.3d linear	221184	1463.52	0.151	1.00	516.92	1.34	n / a
Frozen.3d quadratic		784.04	0.282	1.00	736.12	1.20	n / a
Warm.3d weak scale		741.20	0.298	1.00	700.09	1.21	9.73
Warm.3d strong scale		719.80	0.307	1.00	679.68	1.28	9.45
LWFA - 01 - strong scale		70.91	3.119	4.66	76.55	9.48	16.80

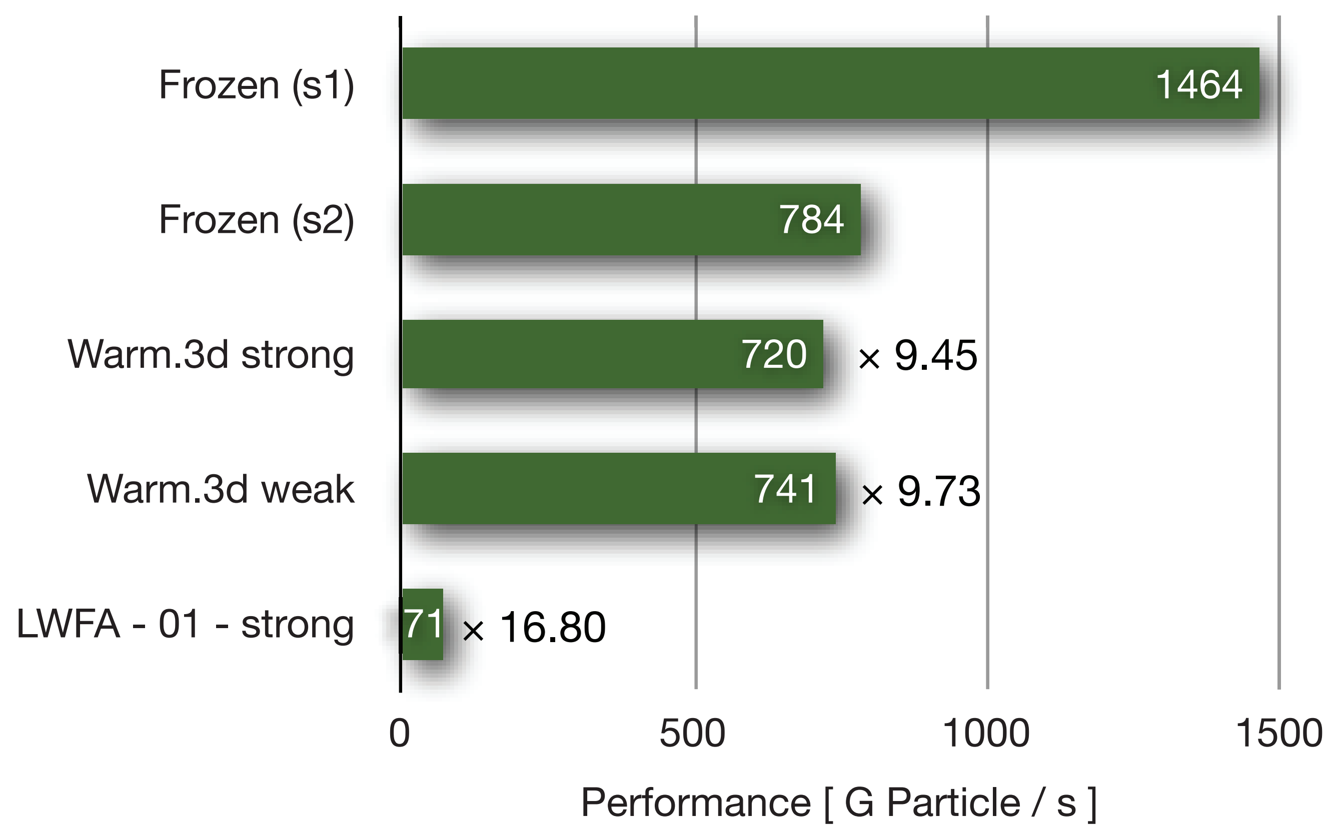


Friday, August 24, 2012

OSIRIS : **Particle Injection in Laser Wakefield**

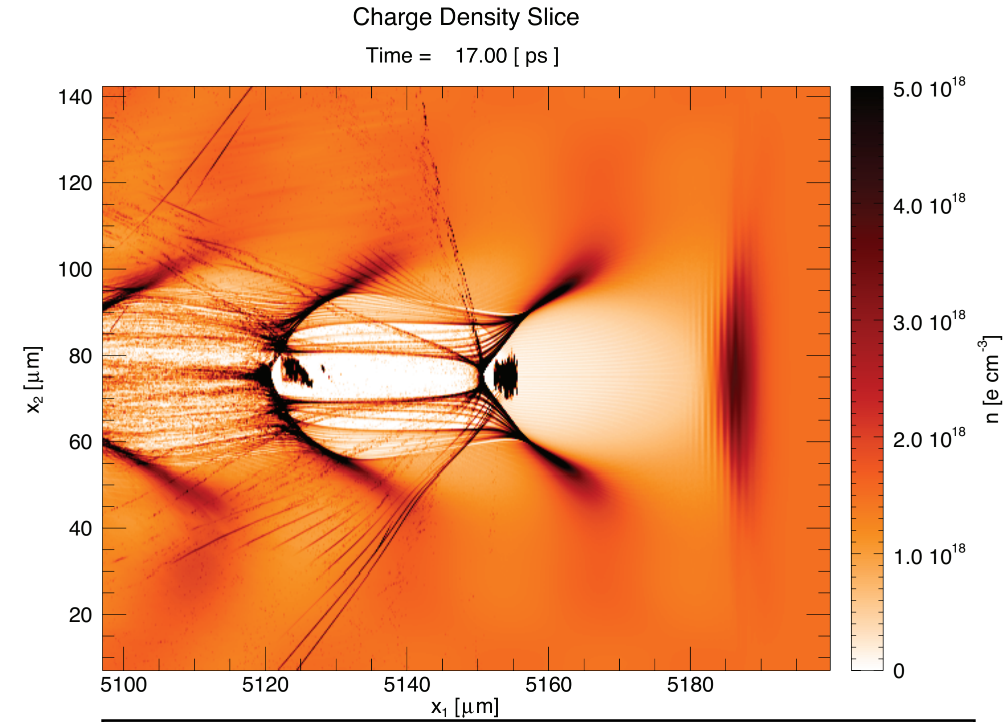
Run	Partition [cores]	Performance [G part/s]	Push Time [μs]	Average Imbalance	TFLOPS	INS/FP	Speedup
Warm.3d	55296	179.95	0.307	1.00	169.92	1.28	2.36
LWFA - 01		29.66	1.864	3.64	31.18	6.39	7.03
LWFA - 02		27.43	2.016	4.75	28.02	7.69	7.37
LWFA - 03		61.20	0.903	2.31	58.25	3.84	6.92
Frozen.3d linear	221184	1463.52	0.151	1.00	516.92	1.34	n / a
Frozen.3d quadratic		784.04	0.282	1.00	736.12	1.20	n / a
Warm.3d weak scale		741.20	0.298	1.00	700.09	1.21	9.73
Warm.3d strong scale		719.80	0.307	1.00	679.68	1.28	9.45
LWFA - 01 - strong scale		70.91	3.119	4.66	76.55	9.48	16.80

221K Algorithm Performance

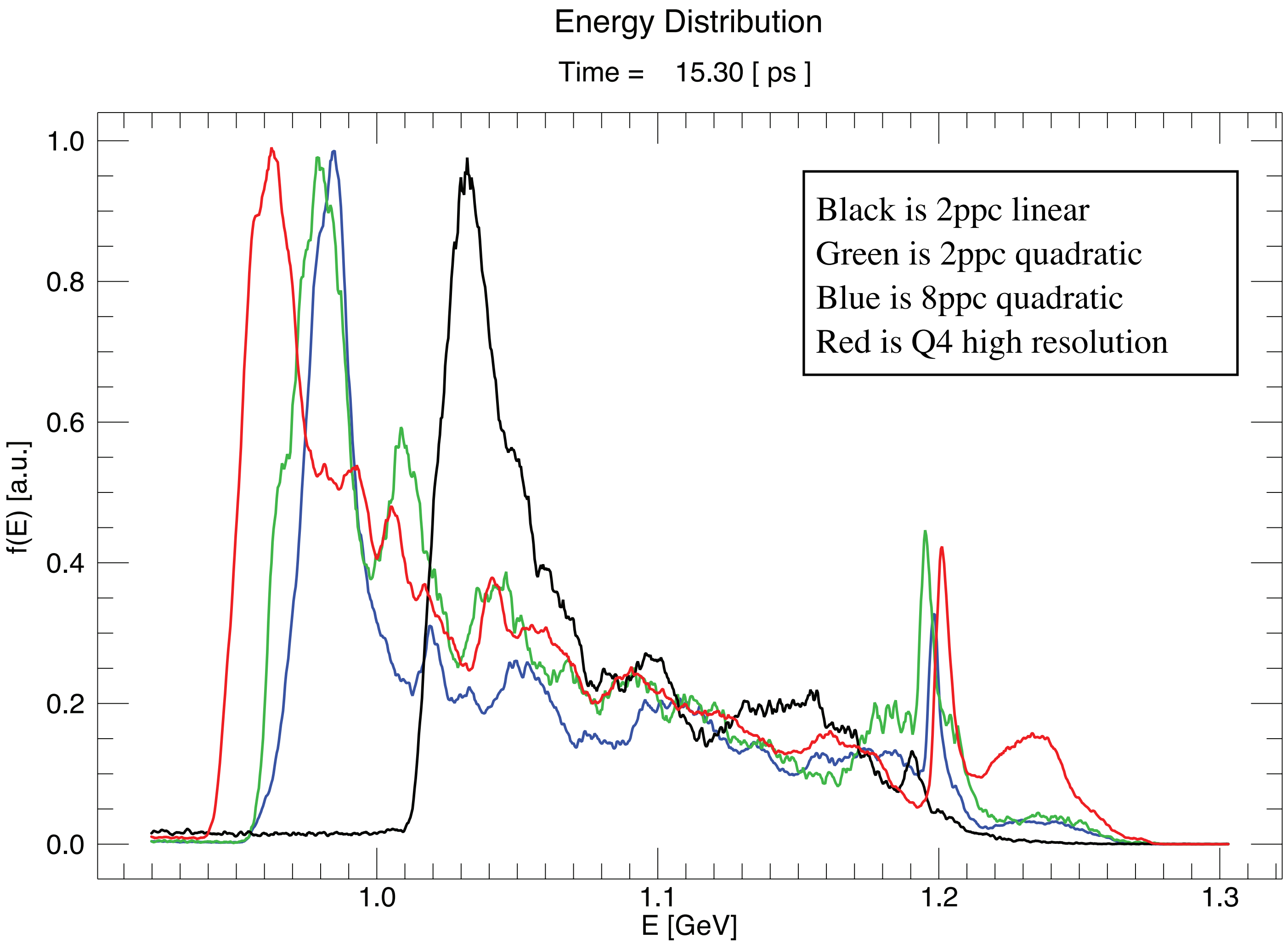


OSIRIS : Particle Injection in Laser Wakefield

	2ppc Linear	2ppc Quad	8ppc Quad	Q4 HR
Charge [pC]	284	339	347	366
Avg. Ene [MeV]	1074.7	1052.6	1054.7	1048.1
StdDev Ene [MeV]	53.4	76.3	75.6	85.5
Peak Ene [MeV	1031.8	979.0	984.5	962.6
Ene FWHM [MeV]	34.5	50.1	19.8	44.5
ϵ_{Ny} mm mr	29.6	26.7	28.8	19.2
ϵ_{Nz} mm mr	33.9	30.7	25.6	18.6



A 2D slice of the electron density showing the electrons injected into the first two buckets.



Comparison of the energy spectra of the beam in the first bucket for the runs.

- Charge (the linear particle shape run has 25% less charge) and the emittance are significantly reduced in the higher resolution (Q4) run.
- The high resolution run has 50% lower RMS value for the two transverse planes.
- This improvement in emittance is very important for both collider and light source applications.

Friday, August 24, 2012

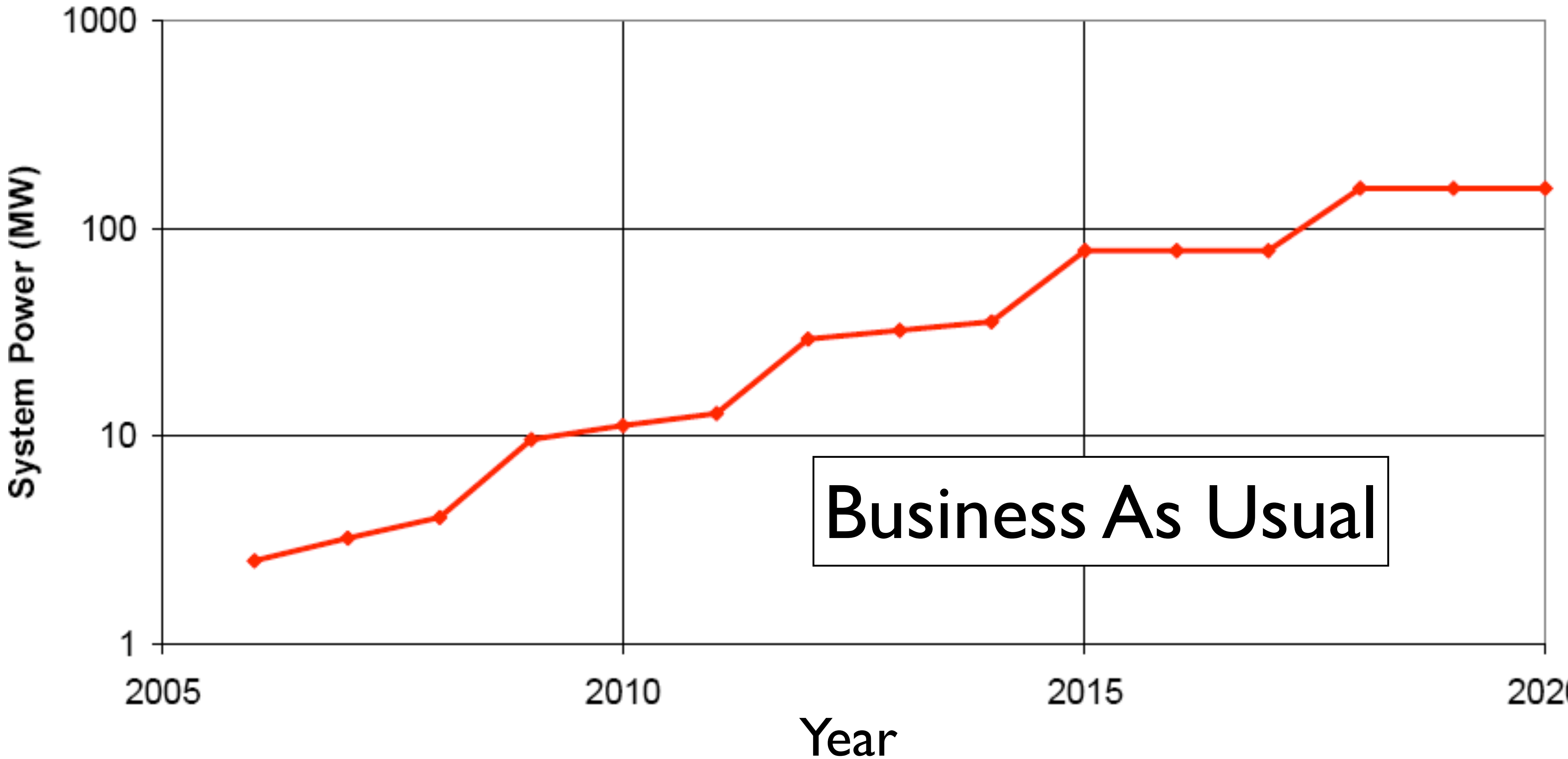
ASCR

- At $1M per MW, energy costs are substantial
- 1 Pf in 2010 ~ 3 MW
- 1 Ef in 2018 at 200 MW with "usual" scaling

NNSA

- Power constraints using current technology are unaffordable
- 20 Pf Sequoia requires ~ 10MW to operate
- 1Ef requires ~500MW with current technologies

1 Exaflop in 20?? at 20 MW is target!



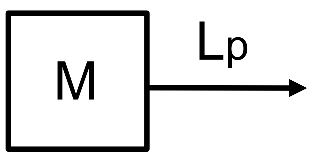
Exascale Table -guess work?

	2010	2018	Factor Change
System peak	2 Pf/s	1 Ef/s	500
Power	6 MW	20 MW	3
System Memory	0.3 PB	10 PB	33
Node Performance	0.125 Gf/s	10 Tf/s	80
Node Memory BW	25 GB/s	400 GB/s	16
Node Concurrency	12 cpus	1,000 cpus	83
Interconnect BW	1.5 GB/s	50 GB/s	33
System Size (nodes)	20 K nodes	1 M nodes	50
Total Concurrency	225 K	1 B	4,444
Storage	15 PB	300 PB	20
Input/Output bandwidth	0.2 TB/s	20 TB/s	100

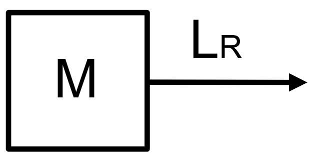
Delivery Date 2020-2022
Performance 1000 PF LINPACK, 300 PF on codesign applications
Power Consumption 20 MW (not including cooling)
MTBAI 6 days (mean time between application interruptions)
Memory including NVRAM 128 PB

Extended Scope of Application Software Problems

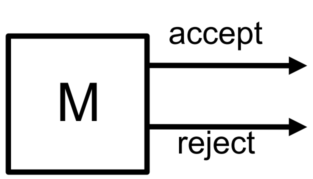
Example Problem: solving algebraically determined systems of linear equations numerically (Linpack TOP500, FLOPs)



• construct A, b given dim(A) dim(A)=dim(rank(1,A))=dim(rank(2,A))



• compute A=LU, solve Ly=b for y, solve Ux=y for x



• compute $|A|_{inf}$, compute $|x|_2$, compute $|b\text{-}Ax|_2$, approximate ε (machine precision), form $\kappa = |b\text{-}Ax|_2 / (|A|_{inf} * |x|_2 * \varepsilon)$; if $\kappa \sim$ O(dim(A)) **accept** x, else **reject** x

Ex2: BFS(Graph500,TEPS)

```
Input: Graph G(V,E), source node s0
Output: Distances from s0 Dist[1..|V|]
∀v ∈ V do:
    dist[v] = ∞
dist[s0] := 0
Q := ∅
Q.enqueue(s0)
while Q ≠ ∅ do
    i := queue.dequeue()
    for each neighbor v of i do
        if dist[v] = ∞ then
            dist[v] := dist[i] + 1
            Q.enqueue(v)
        endif
    endfor
endwhile
```

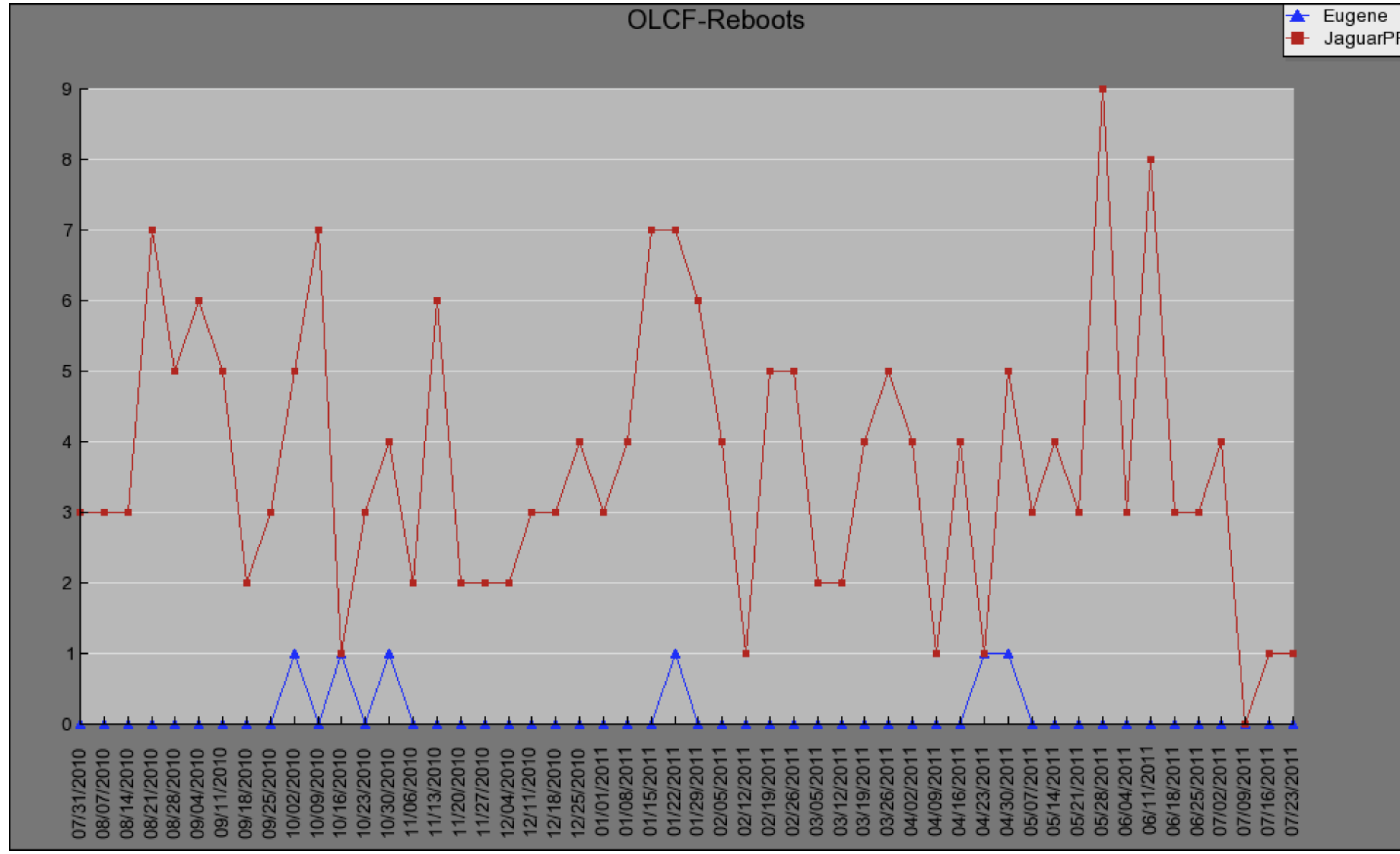
Q: How do the language of the problem and the accepted result relate to reality?
Requires analysis beyond software analysis above and distinguishes ***computational science*** from system and library software development. Takes more time -needs refinement phase of algorithms and metrics.

Metric: the distance between two points in some topological space

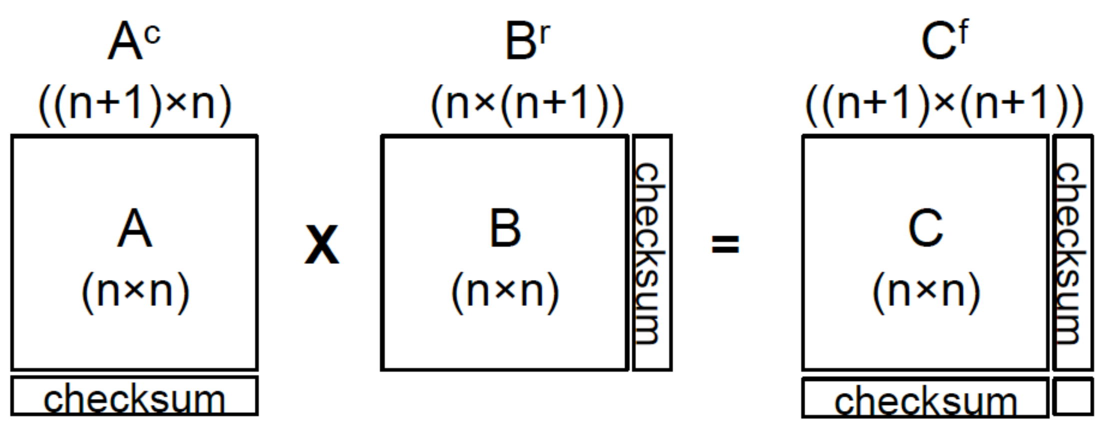
Challenge: detecting, mitigating, recovering from failures

- fail / continue
- hard / soft faults

- resiliency must go beyond check point / restart
  - algorithm based fault tolerance



have to go beyond single failure



$$\mathbf{A}^c_{n+1,j} = \sum_{i=1}^{n} \mathbf{A}_{ij}$$

$$\mathbf{B}^r_{i,n+1} = \sum_{j=1}^{n} \mathbf{B}_{ij}$$

$$\mathbf{C} = \mathbf{A} * \mathbf{B} \text{ and } \mathbf{C}^f = \mathbf{A}^c * \mathbf{B}^r$$

$$\mathbf{C}^f_{n+1,j} = \sum_{i=1}^{n} \mathbf{C}^f_{ij} \quad \mathbf{C}^f_{i,n+1} = \sum_{j=1}^{n} \mathbf{C}^f_{ij}$$

Challenge: quantify the data related costs on and across nodes

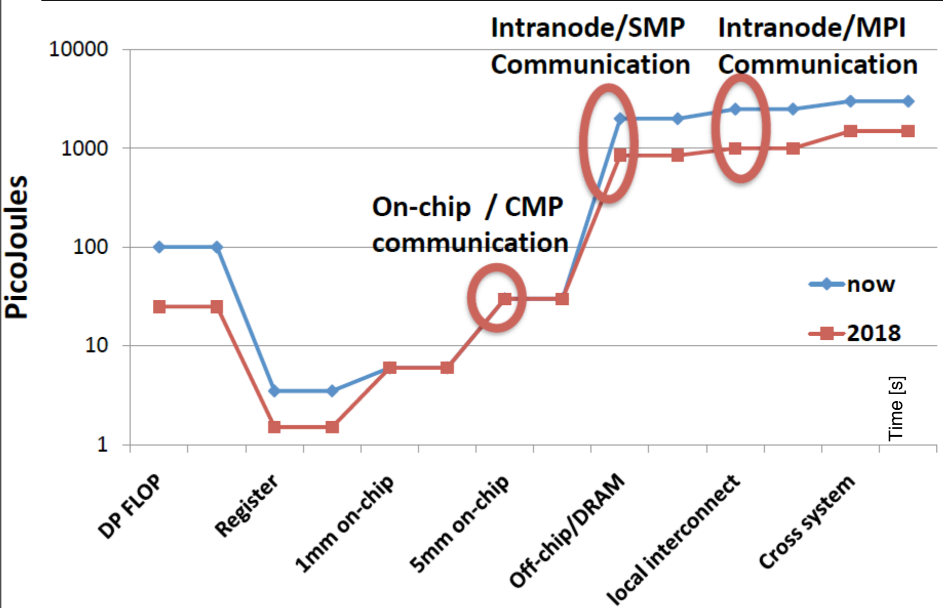
-refine performance measures for data movement and access costs as these dominate over floating point costs

- ***bandwidth***, the number of cycles a core waits because the bus is not ready; as the measure gets large, it indicates that the bus is in high demand and loads or stores involving main memory will take longer

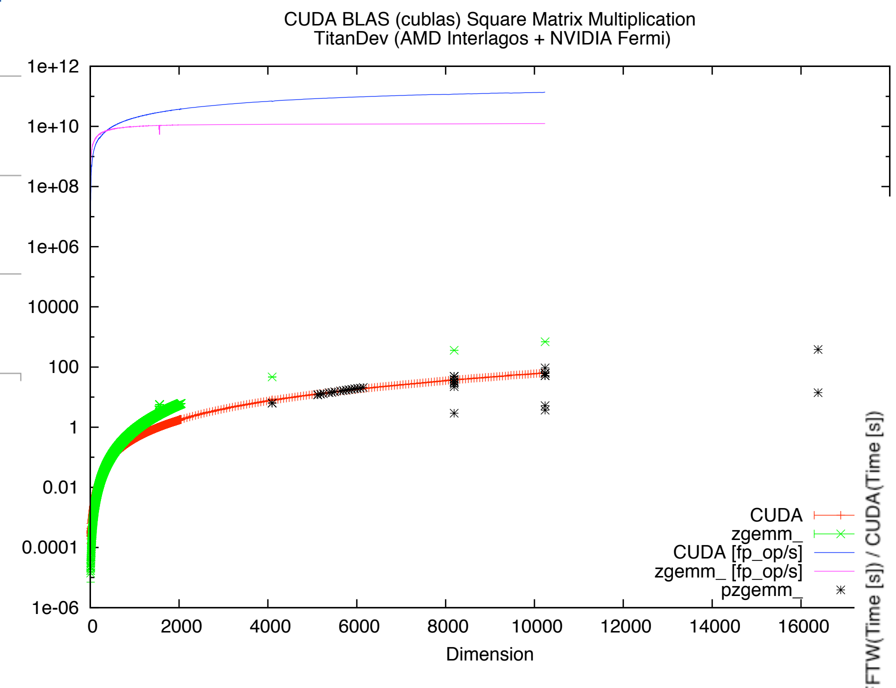
  -provides means to reason about performance costs versus (bisection) bandwidth scaling (i.e. increased node counts)

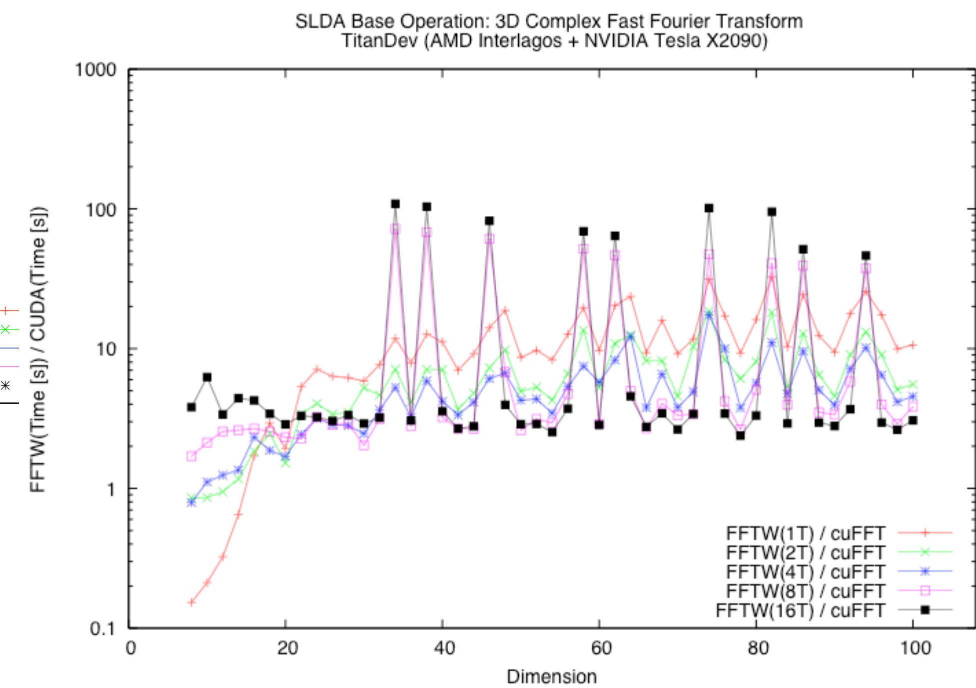
- ***locality***, the ratio of the peak versus measured capacity of each memory level (on/off chip) divided by access time in cycles

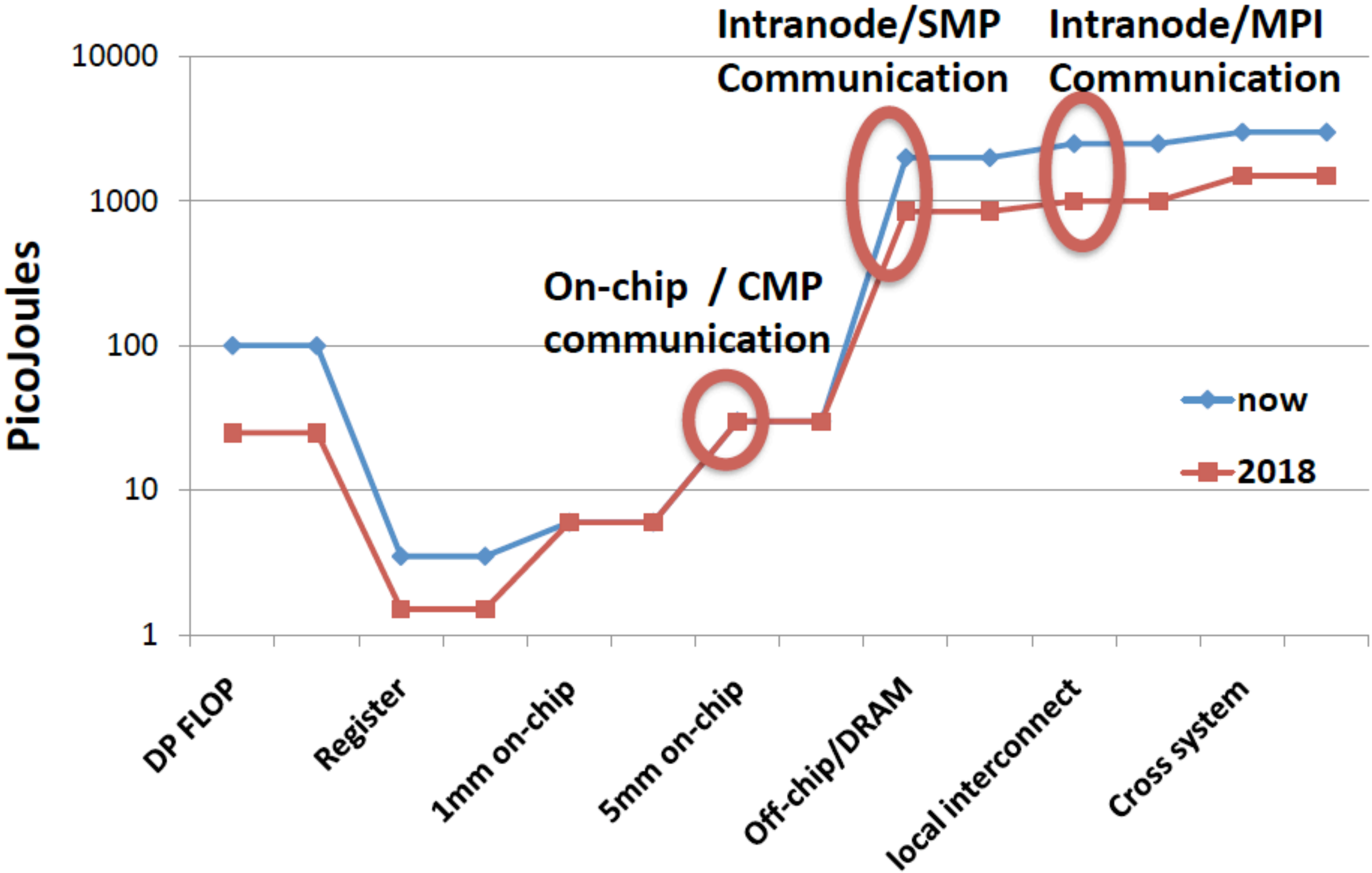
- i.e. consider ratio of gather and scatter costs in loops (A. Snavely, exascale planning meeting)



ref. ASCR exascale mtg







ref. ASCR exascale mtg

Need extensions that relate performance to power; lead to novel optimization ideas

-extension of existing metrics to reason about power and performance tradeoffs, energy driven optimizations (i.e. DVFS)

-number of floating point operations per Watt (floating point dominated)

-cost of loads or stores in bytes per Watt (data ops dominated)

-metric guided optimizations to simultaneously minimize power consumption and time to solution (IBM Zurich study)

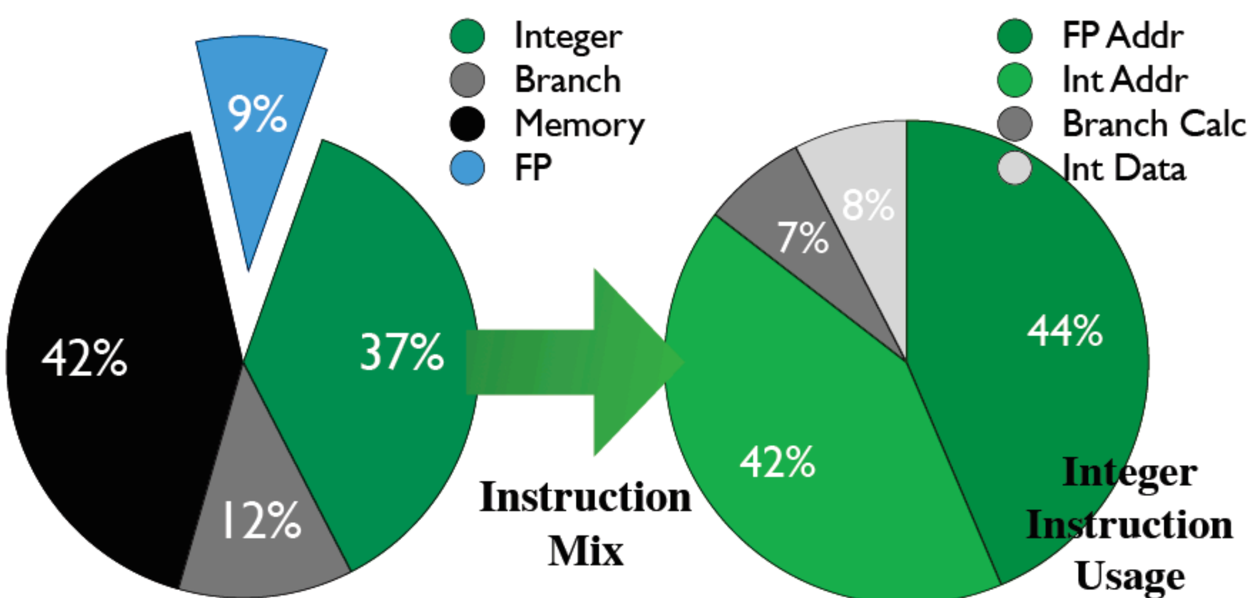
-computational cost ~ *f(time to solution) * energy*

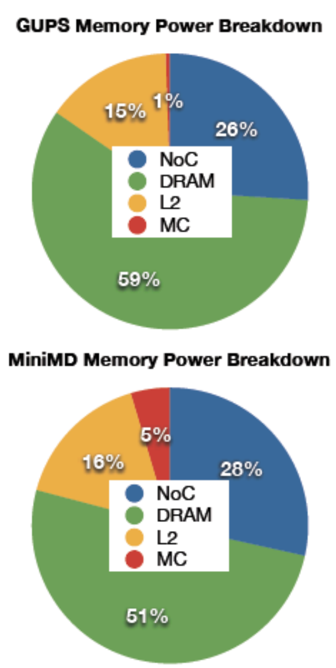
-*f* constant, cost per execution event in Joules

-*f* linear, cost provides insight about appropriateness of hardware platform for application

-demand tools for power measurements

-memory (29%), network (29%), floating point unit (16%)) (distribution of power in HPC hardware (Kogge))



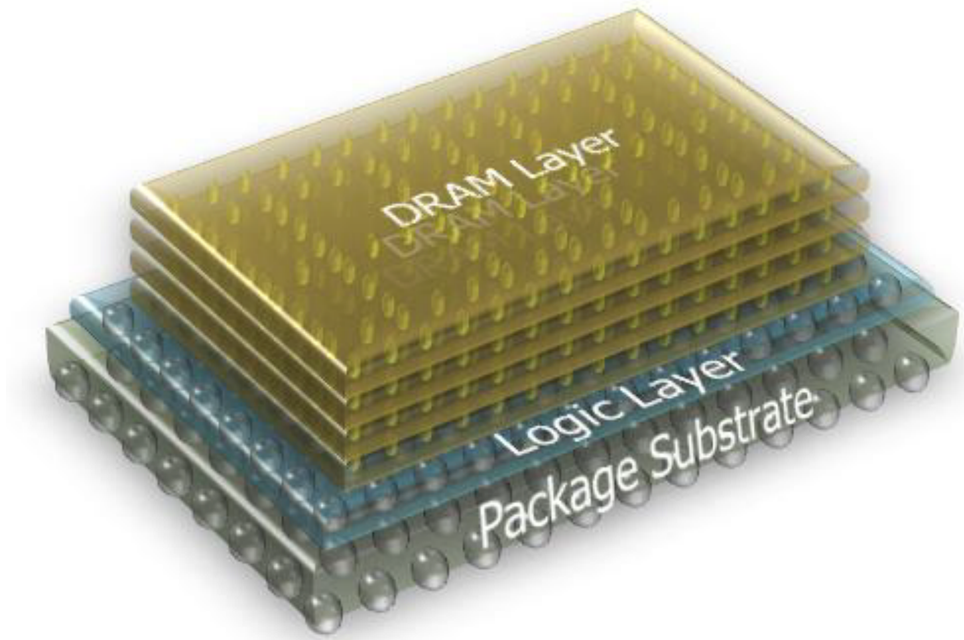


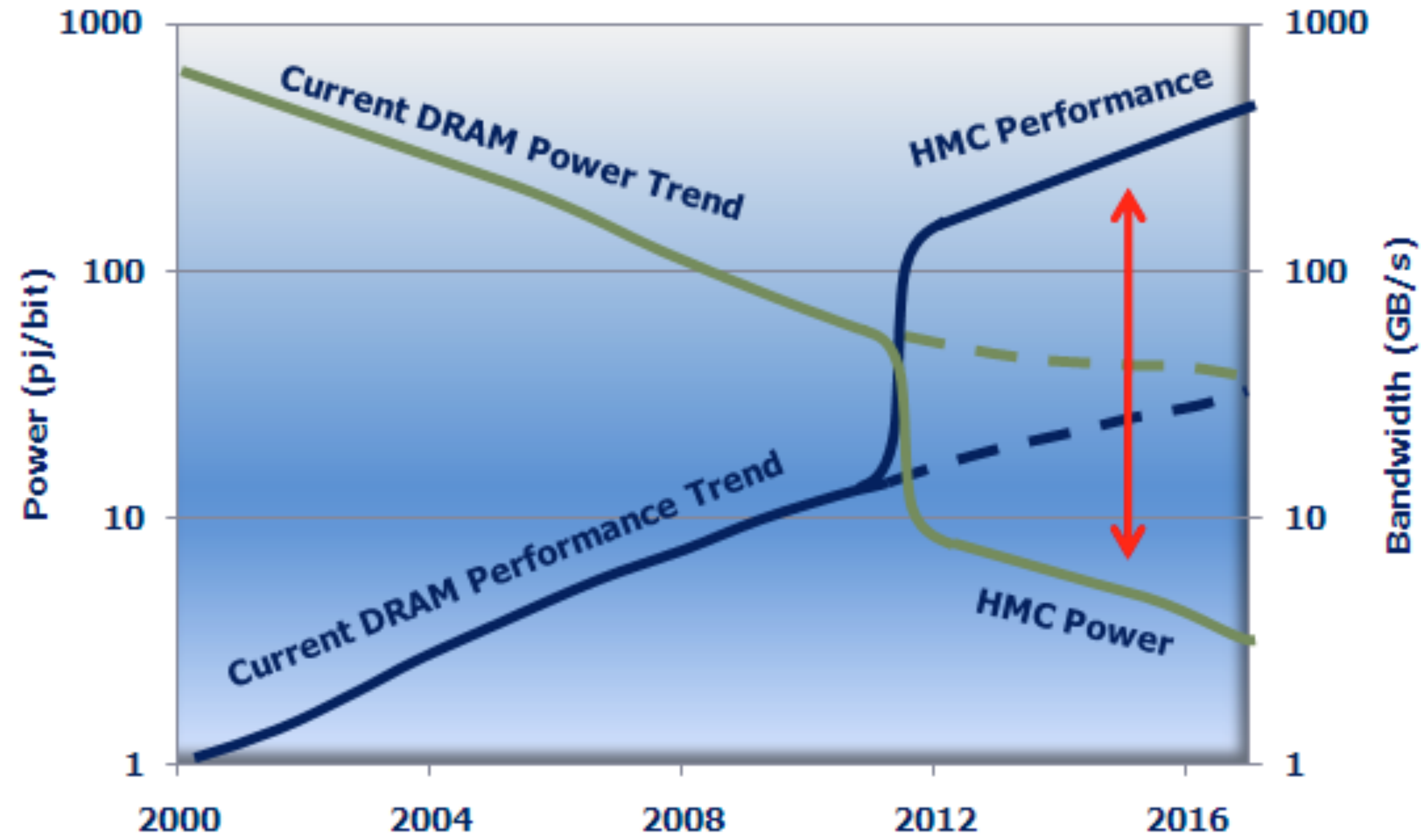
-relate cycle costs in memory refs to energy in Joules

To Where	Cycles
Register	≤ 1
L1d	~ 3
L2	~ 14
Main Memory	~ 240

ref. ASCR exascale mtg

Micron
Hybrid Memory Cube





- Current DRAM roadmap will not enable achieving exascale systems with anything like the expected needs and goals

Reduced latency – With vastly more responders built into HMC, we expect lower queue delays and higher bank availability, which can provide a substantial system latency reduction, which is especially attractive in network system architectures.

Increased bandwidth — A single HMC can provide more than 15x the performance of a DDR3 module. Speed is increased by the very fast, innovative interface, unlike the slower parallel interface used in current DRAM modules.

Power reductions — HMC is exponentially more efficient than current memory, using 70% less energy per bit than DDR3.

Smaller physical systems — HMC's stacked architecture uses nearly 90% less space than today's RDIMMs.

Pliable to multiple platforms — Logic-layer flexibility allows HMC to be tailored to multiple platforms and applications.

Exascale System Networks

REQUIREMENTS

Scale

100,000 –1,000,000 nodes

Node Bandwidth

10 GB/s –2000 GB/s

Very application dependent

Power efficiency

Particularly important for HPC

Latency

Critical for HPC systems

CHALLENGES

Interconnect density

Chip edge, board edge, enclosure

Low Network Diameter

Benefits latency, power and reliability

Requires high radix switches

Cabling complexity

Particularly with low diameter networks

POWER CHALLENGE

Total BW = Nodes x BW x hops x bit

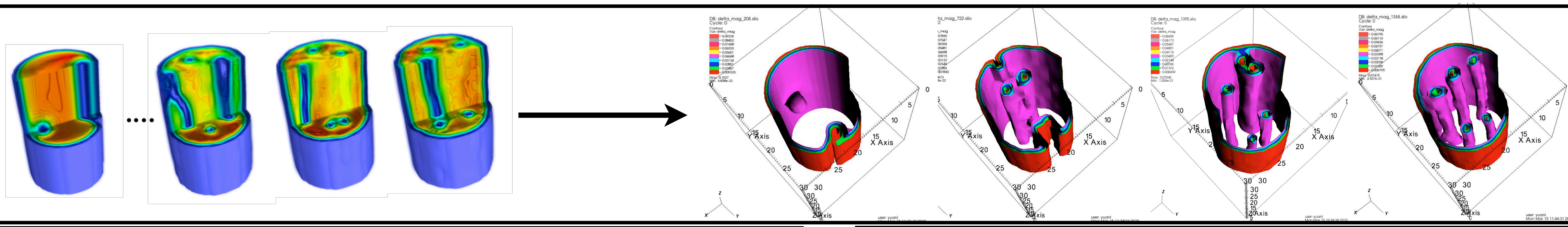
= 100,000 x 2000 x 4 x 8

= 6.4 Exabits / s

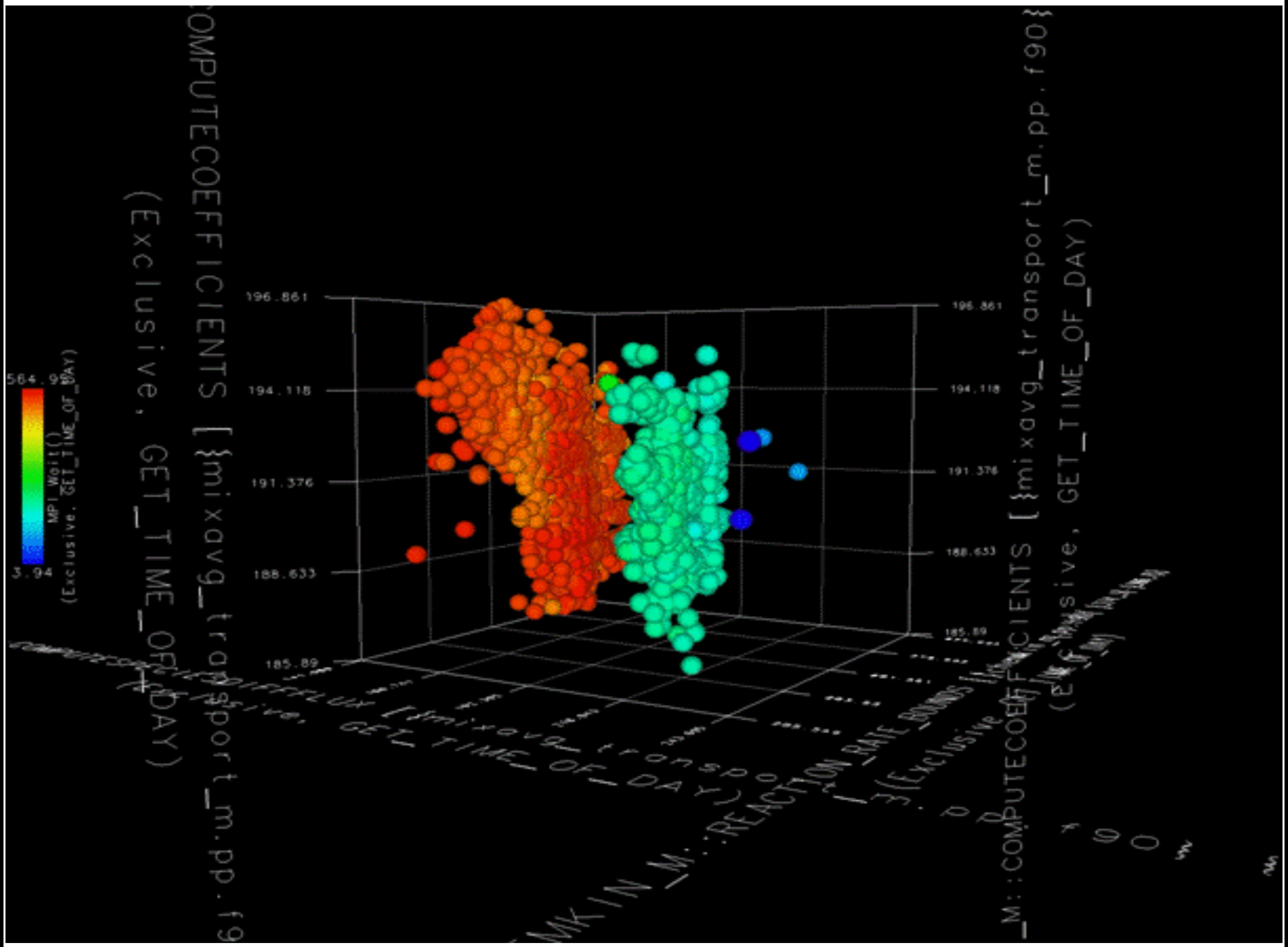
Power = 30MW

4.7pJ/bit available entire power budget for interconnect!

Challenge Tools to Pinpoint Performance and Numerical Errors, Drive Science Based Feature Extraction in Massive, Complex Data Sets

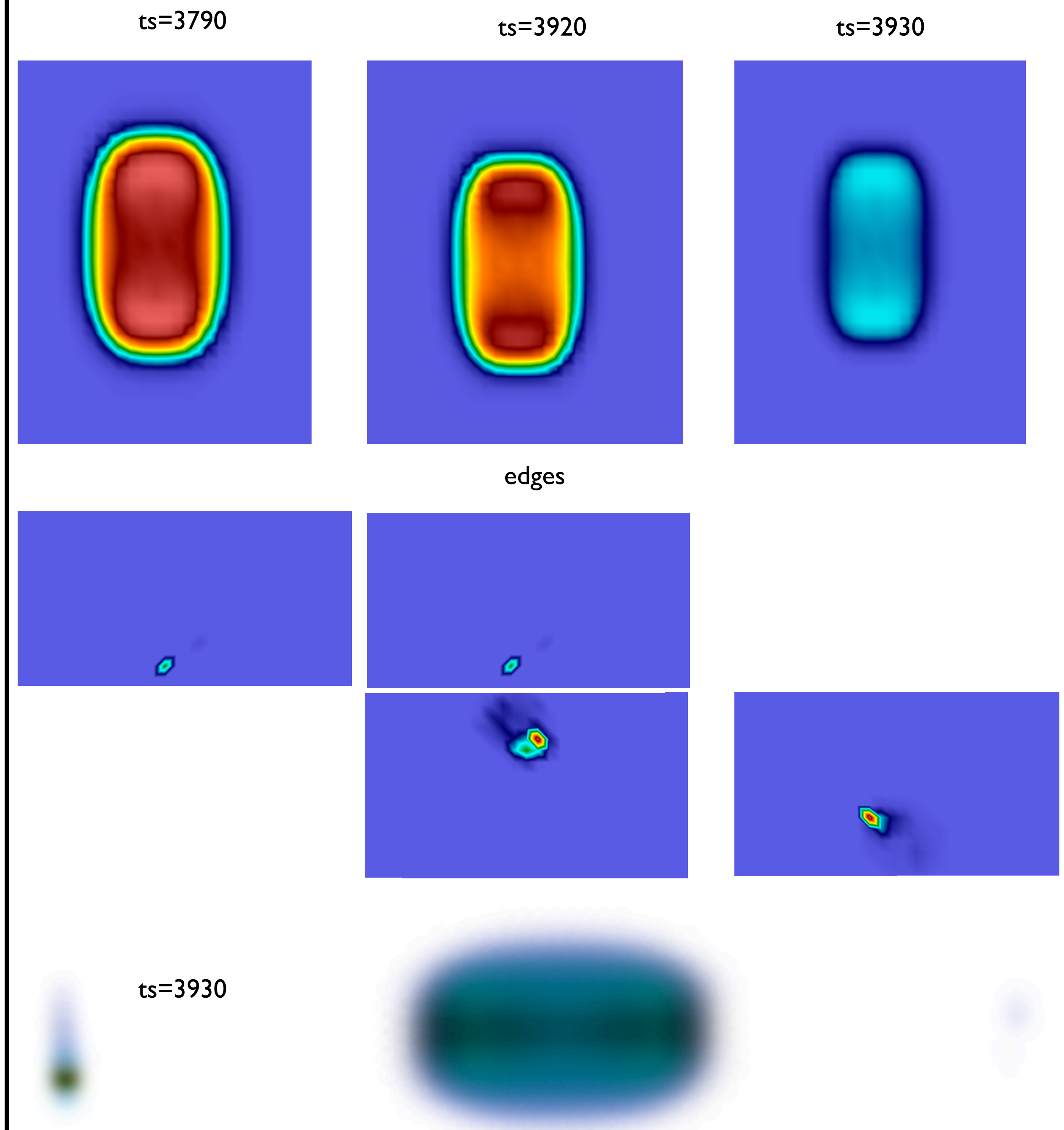


Machine Events Are Useful But Cannot Tell Whole Story



This problem completed execution successfully from the application software perspective.

There is a clear problem in the performance.



Challenge accurate, scalable tools at *thread level*

	Multiplies	Adds	Total
real	mnl + 2mn	mnl	2mnl + 2mn
complex	4mnl + 8mn	4mnl + 4mn	8mnl + 12mn

Table 14: Theoretical complexity of $C(m,n) \leftarrow \alpha A(m,l)B(l,n) + \beta C(m,n)$.

Problem PEs nt/PE m l n chnk	FP	INS	L2DCM	Time[μs]
2 pes, 4nt/pe 1024,1024,1024,256	init()	init()	init()	init()
p0,t0	3145728	123983711	2089784	2422204
p0,t1	3145728	126814035	2107375	2421820
p0,t2	3145728	107087054	2124844	2421705
p0,t3	3145728	107498702	2100952	2421780
p1,t0	3145728	144025387	2189125	2541868
p1,t1	3145728	147571937	2220183	2541458
p1,t2	3145728	107456375	2214333	2541361
p1,t3	3145728	109273232	2200654	2541429
1024,1024,1024,256	work()	work()	work()	work()
p0,t0	2151153664	7780969482	270106544	11254443
p0,t1	2151153664	9020932602	270484334	11254098
p0,t2	2151153664	7525985513	270171124	11254286
p0,t3	2151153664	9273025751	270499158	11254282
p1,t0 THY	2151153664	7628607535	270355014	12324730
p1,t1	2151153664	9077245107	270414465	12324461
p1,t2	2151153664	7525968734	270386539	12324582
p1,t3	2151153664	9337961638	270389136	12324478
Totals	17234395136 17205035008	68144406795	2180053564	14866598

Table 17: Measured machine events of threaded parallel work phase (zgemm).

1 PE, 4 nt / PE
Group / Function / Thread (max)
==============================
Total

Time% 100.0%
Time 12.213947 secs

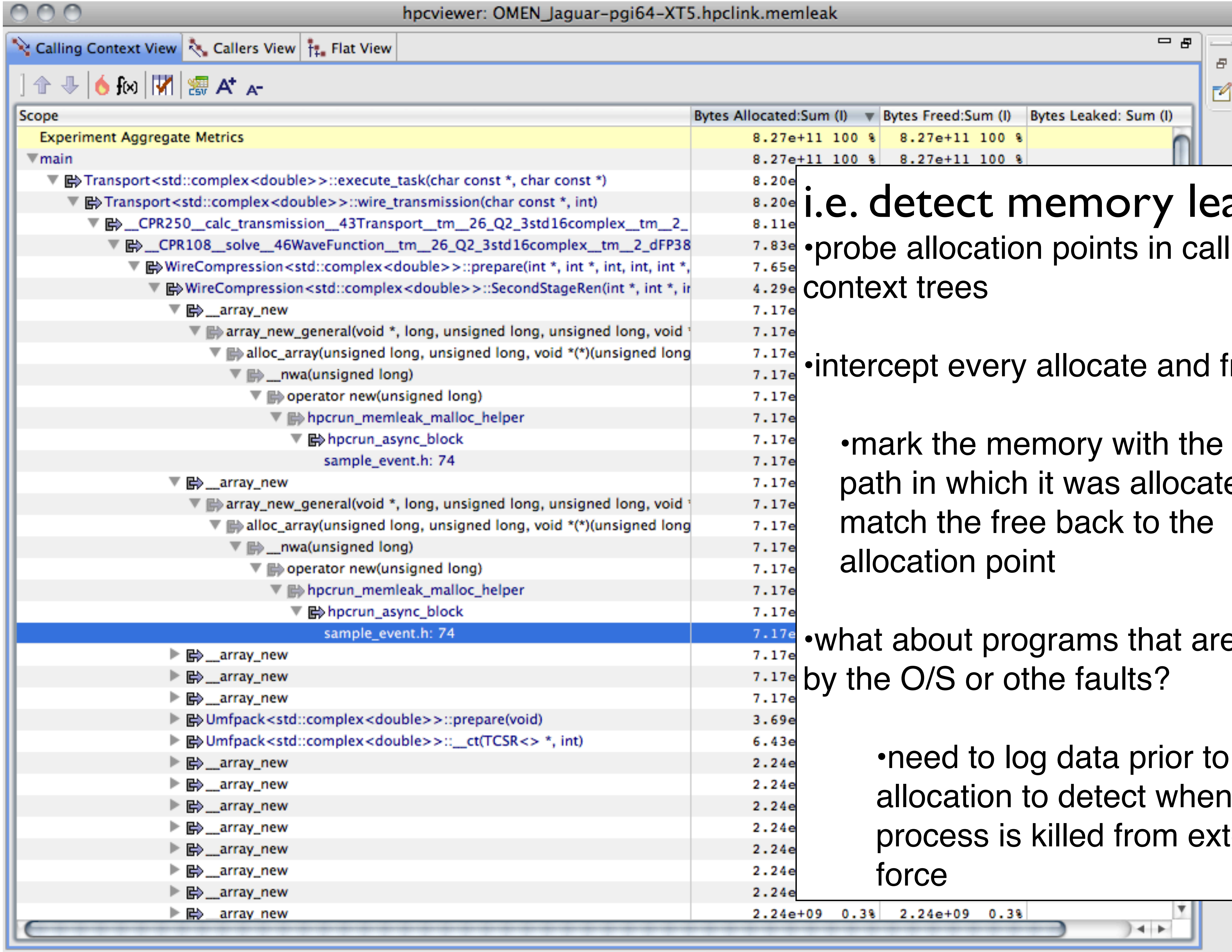
TOT_INS
1037.779M/sec
10063040357 instr

FP_INS
222.330M/sec
2155872263 ops (2154299392)

TOT_CYC
9.697 secs
21332826724 cycles

User time (approx) 100.0% Time
12.214 secs
26870760748 cycles

Challenge accurate, scalable memory tools

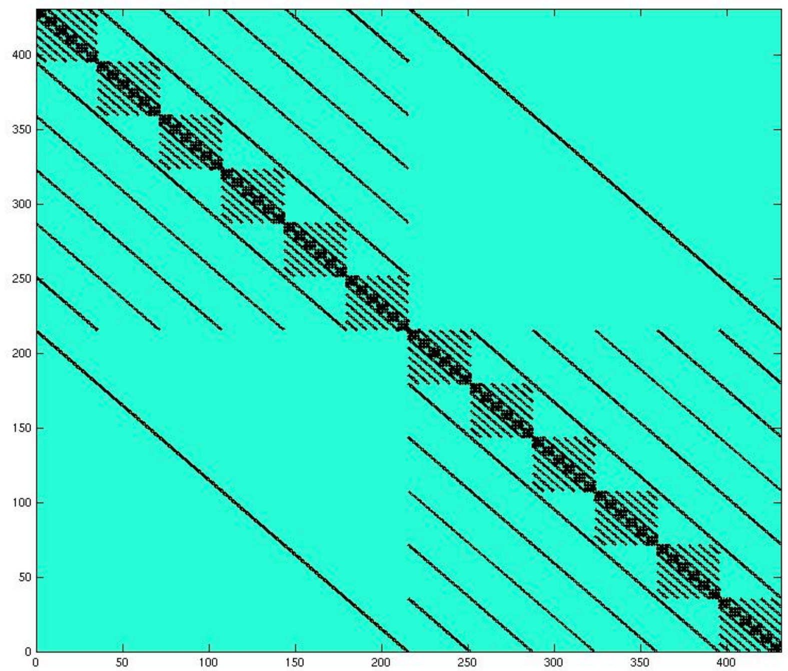


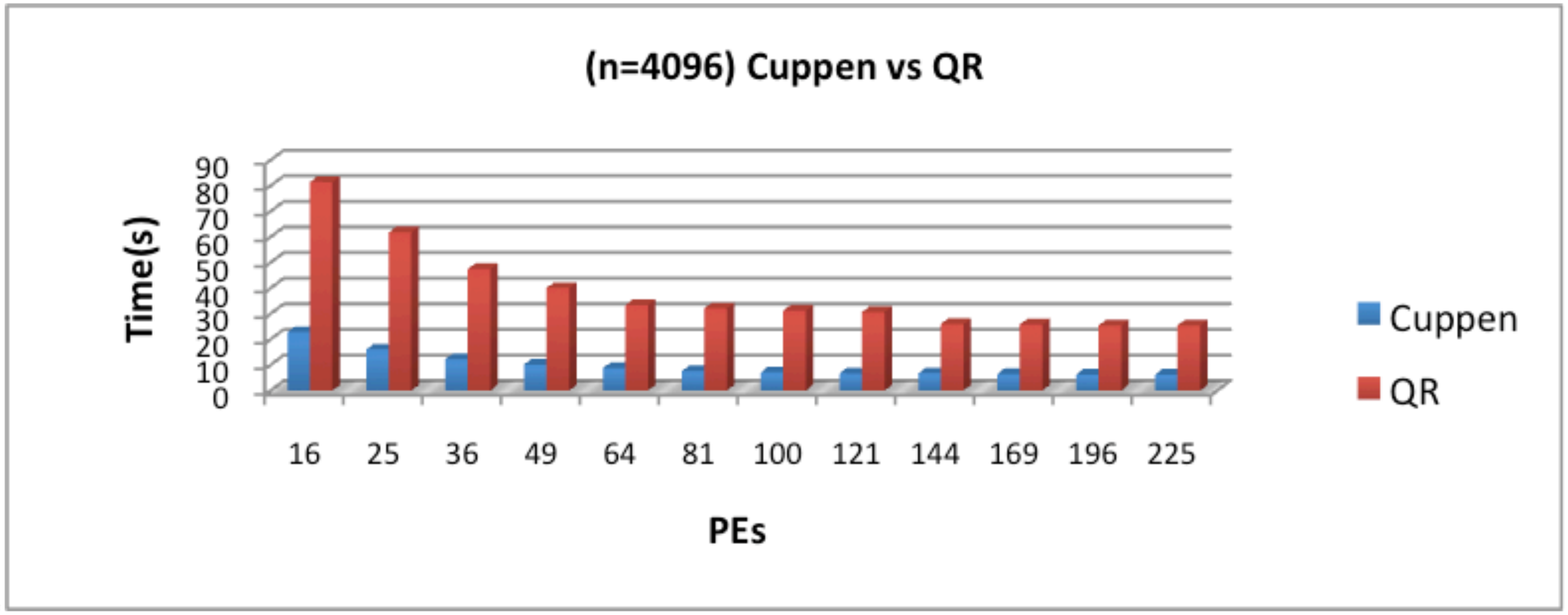
i.e. detect memory leaks

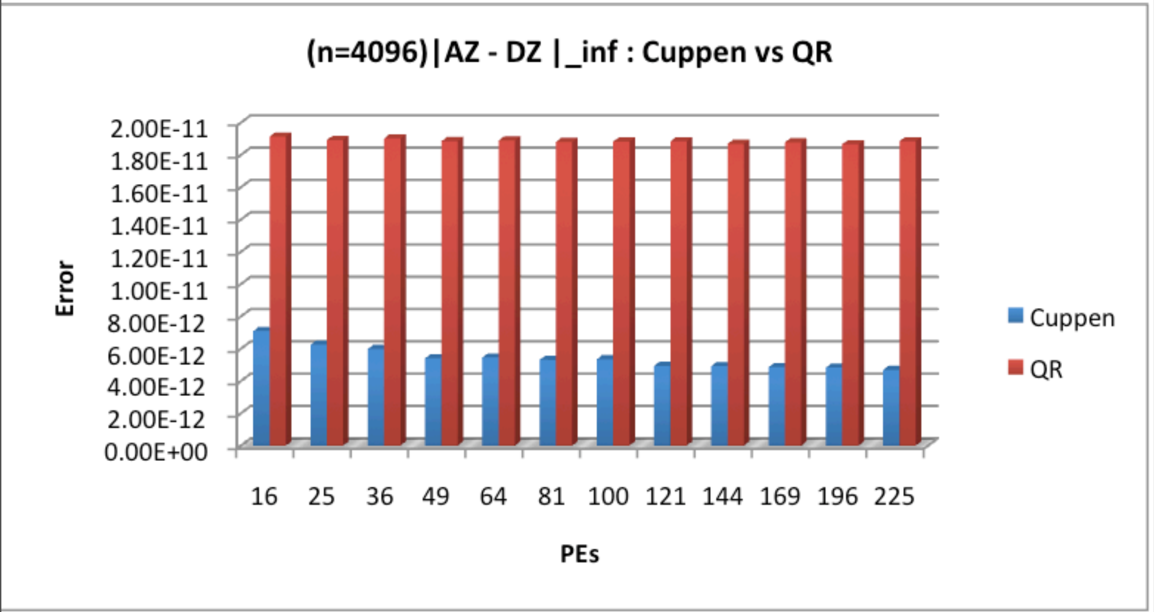
- probe allocation points in calling context trees
- intercept every allocate and free
    - mark the memory with the call path in which it was allocated, match the free back to the allocation point
- what about programs that are killed by the O/S or othe faults?
    - need to log data prior to allocation to detect when a process is killed from external force

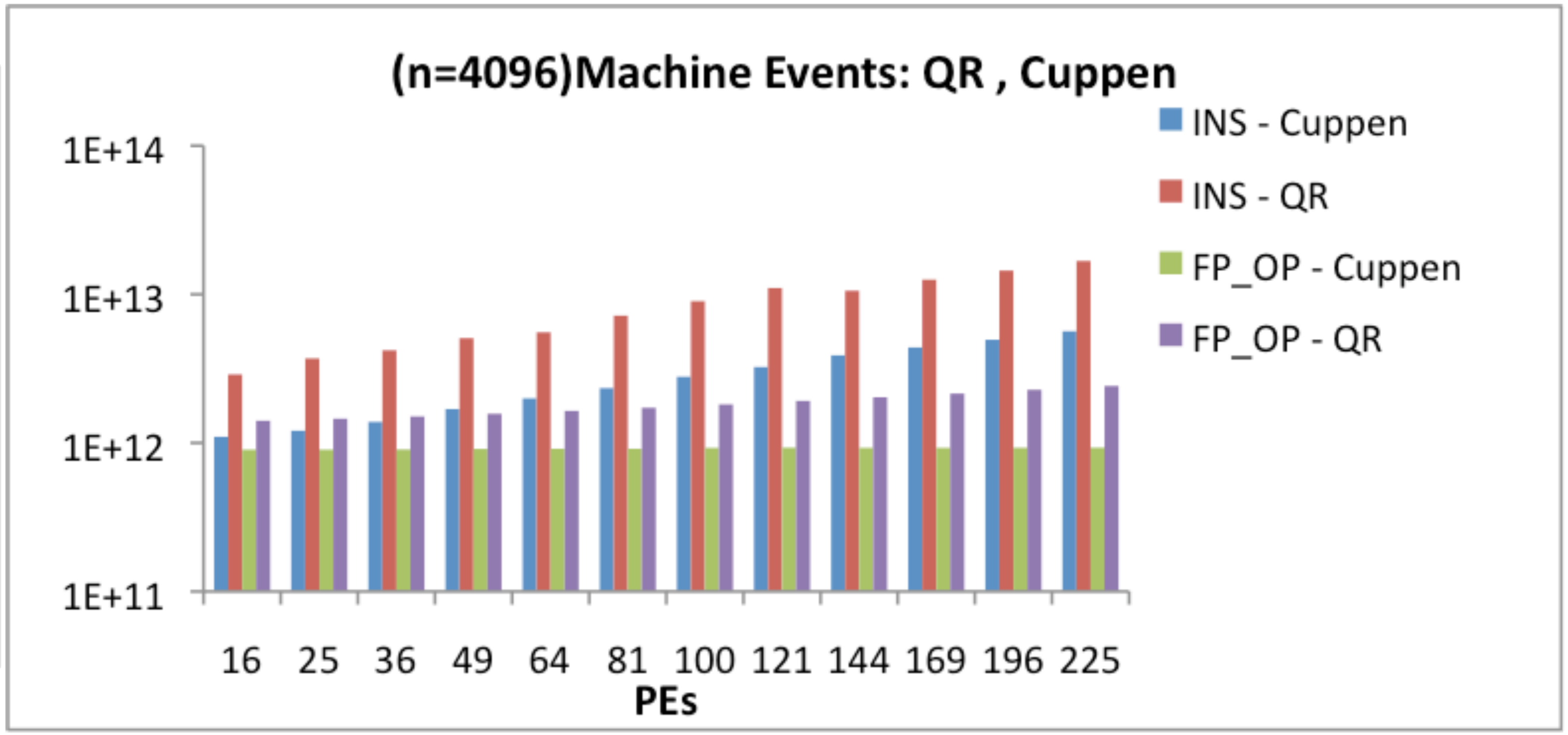
Challenge algorithms that Improve {ins,*flop(s)*} / *byte* (and don't compromise accuracy or performance)

•J.J.M. Cuppen, *A Divide and Conquer Method for the Symmetric Tridiagonal Eigenproblem*, Numer. Math. 36, 177-195 (1981)
•F. Tisseur and J.J. Dongarra, *Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem on Distributed Memory Architectures*, lawn132 (1998)





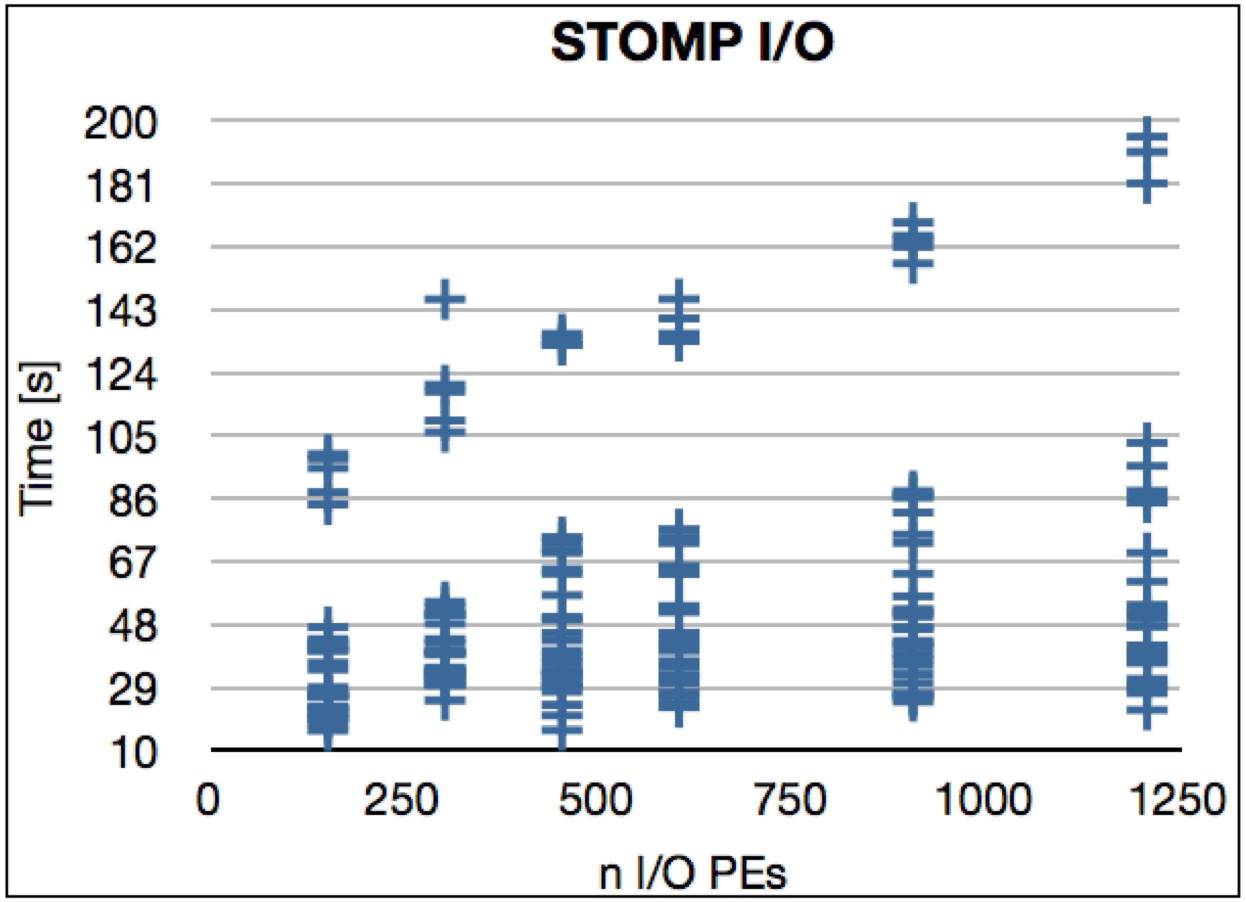


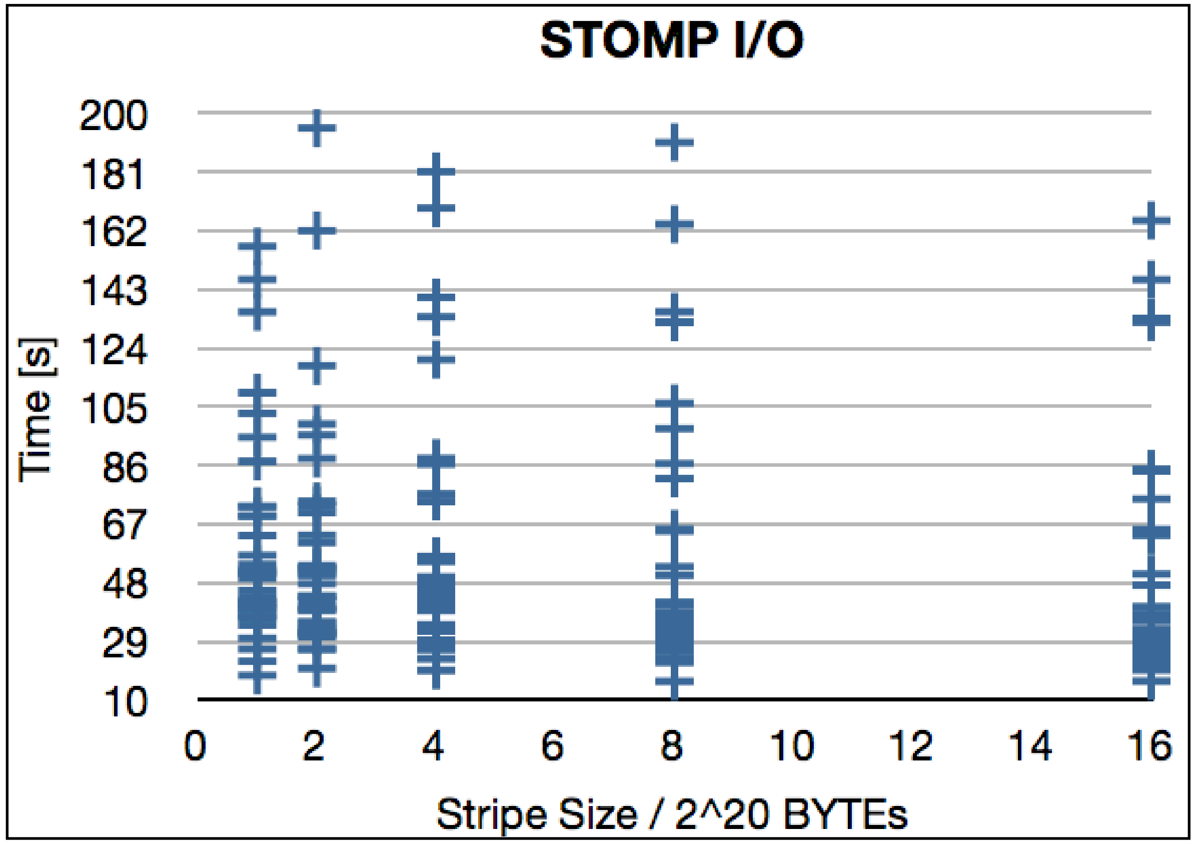


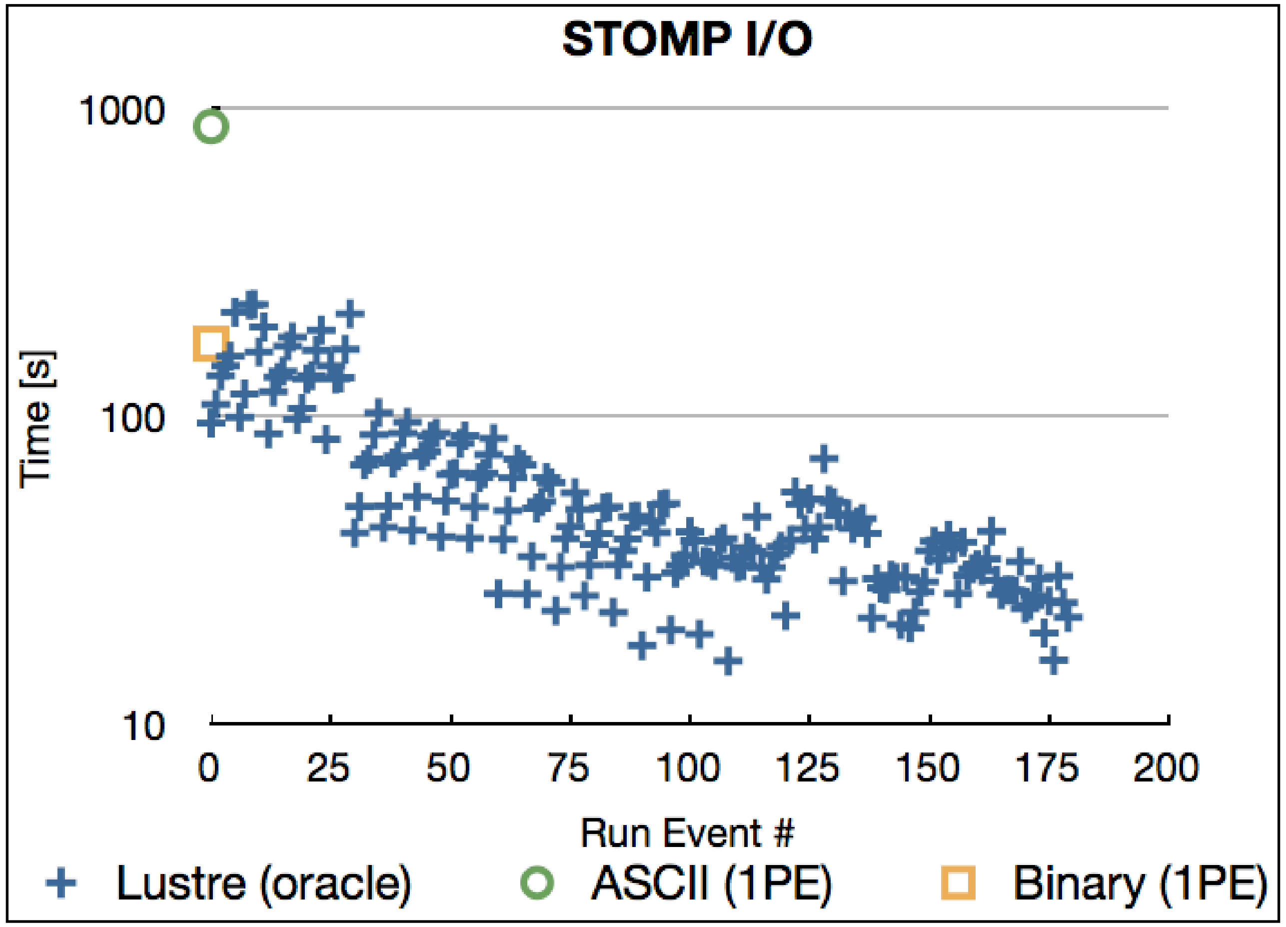
Challenge algorithms that improve I/O operations for applications

Parameters set in the file system related to but independent from the problem parameters:

- Number of OSTs
  1, 2, 4, 8, 16, 32
- Stripe size in BYTEs
  1 MB, 2 MB, 4MB, 8 MB, 16 MB
- access pattern (round robin)
- Number of I/O PEs for spatial decomposition
  kio ~ 1, 2, 3, 4, 6, 8
- Total number of I/O PEs is kio * nfld
  since nfld =151, 151, 302, 453, 604, 906, 1208







Aside on FILEs and IO

ANSI C
- stream of BYTEs
- points to a FILE structure
- fopen,fwrite,fread,fclose

```
void f_copn_ ( char * ffn , int * ffd , int * len ) ;

void f_ccls_ ( int * ffd ) ;

void f_crm_ ( char * ffn , int * len ) ;

void f_cwr_ ( int * ffd , void * fbf , int * fsz , int * nobj , int * ierr ) ;

void f_crd_ ( int * ffd , void * fbf , int * fsz , int * nobj , int * ierr ) ;
```

```
typedef struct  {
        int              level;      /* fill/empty level of buffer */
        unsigned         flags;      /* File status flags          */
        char             fd;         /* File descriptor            */
        unsigned char    hold;       /* Ungetc char if no buffer   */
        int              bsize;      /* Buffer size                */
        unsigned char    *buffer;    /* Data transfer buffer       */
        unsigned char    *curp;      /* Current active pointer     */
        unsigned         istemp;     /* Temporary file indicator   */
        short            token;      /* Used for validity checking */
}       FILE;
```

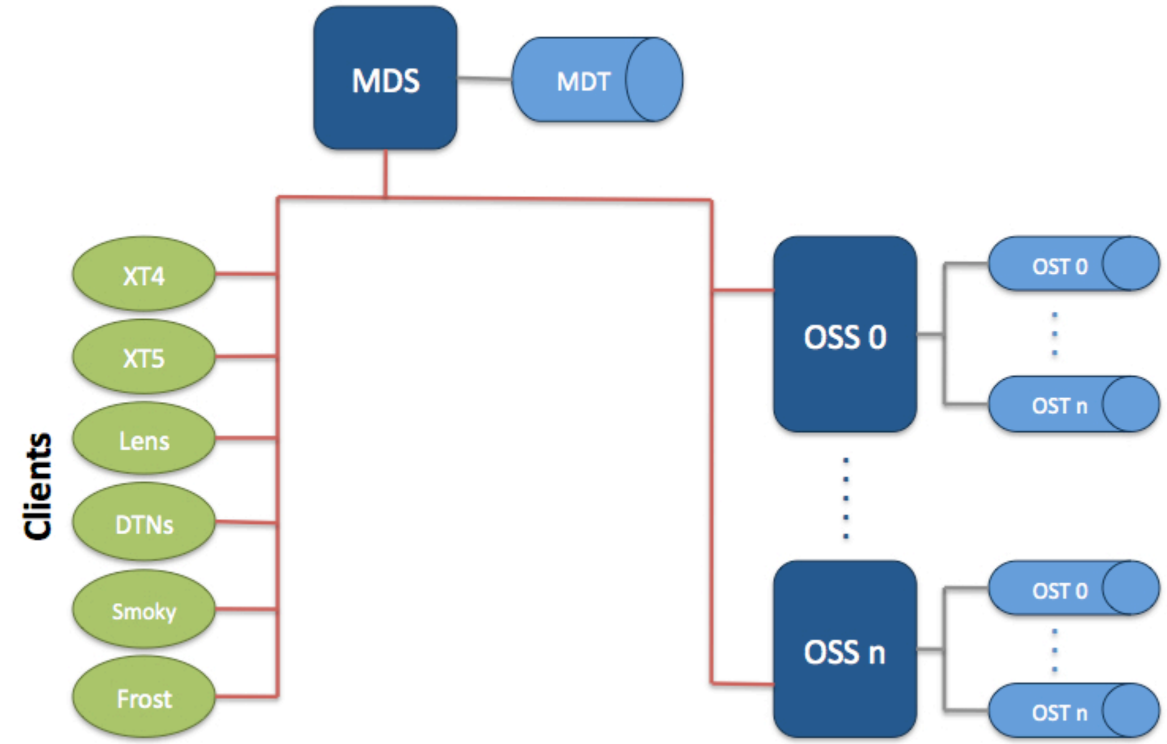
```
0 -rw-r--r-- 1 roche roche 1608 2010-06-21 21:03 fortran-dat.bn
0 -rw------- 1 roche roche 1600 2010-06-21 21:03 c-data.dat
```

Fortran
- sequence of records
- open,write,read,close
- IOLENGTH , RECL

```
fn = '/tmp/work/roche/mpt-omp/ben.txt'//
CHAR(0)

call f_copn ( fn , fd , LEN( fn ) )

call f_cwr ( fd , a , 16 , ndim , ierr )

call f_ccls ( fd )

call f_copn ( fn , fd , LEN( fn ) )

call f_crd ( fd , a_bk , 16 , ndim , ierr )

call f_ccls ( fd )

call f_crm ( fn , LEN( fn ) )
```

Aside on FILEs and IO (2)

POSIX (UNIX)
•stream of BYTES
•file descriptors
  -index into file descriptor table
  -kept in user process
  -points to entry in system in-memory inode table
•open,write,read,close, ioctl



Spider (Lustre) :

•MDS, file names and directories in the filesystem, file open, close, state mgt

•OSS, provides file service, and network request handling for set of OSTs

•OST, stores chunks of files as data objects -may be stripped across one or more OSTs
-Spider has 672 OSTs
-7 TB per OST
-1 MB Default stripe size
-4 Default OST count

Aside on FILEs and IO (3)

```
module load liblut ; -LUT

lut__open() ;

lut__close() ;

lut_putl() ;

pwrite() ;

pread() ;
```

- form modulo classes from MPI communicator over the number of I/O groups
  - for both proton and neutron communicators in nuclear case (44 for protons, 44 for neutrons)

- fit the stripe size to the largest single data item if possible
  - eg for nuclear code and 32^3 lattice, a single 4-component term is 4 * 32^3 * 16 / 2^20 = 2MB

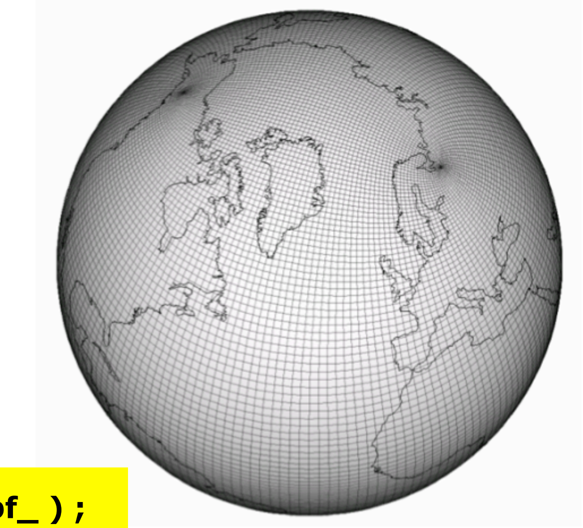
- set the stripe pattern (I use round-robin) and number of target OSTs (I use 88 in nuc code) for target PATH / FILE
  - eg lfs setstripe /tmp/work/roche/kio -s 2m -i -1 -c 88

Performance: POSIX ~ [225,350]MBps , use of Lustre ~ [2,25]GBps

Aside on FILEs and IO (4) - Search Approach

- introduced set of parallel I/O processes within the MPI group
  - (was) gather to single process, followed by sequential write / wait phase within a loop over fields (1 PE writes, nPEs - 1 PEs wait) x nFIELDS iterations
  - (is) loop over (disjoint target) gathers to a set of designated IO PEs; after gather phase then (nIOPEs write in parallel, nPEs - nIOPEs wait) x 1 since nIOPEs > nFIELDS (8 (3D fields / day) × 42 (k-values / fields) × 1 (PE / k-value) = 336 IOPEs / day; 19 IOPEs / day for 2D fields)
- use of lut_putl() library function explicitly invoking LUSTRE file system semantics
- oracle code to search for preferred LUSTRE parameters: number of OSTs, stripe size, number of writers
- similar enhancements for 2D fields; movies require an additional index transformation which is done locally by the IO PE prior to writing (block cyclic to natural column major)

```
memcpy( ( void * ) fnbf , ( const void * ) ffn , ( size_t ) *len ) ;
for ( iniopes = 0 ; iniopes < 6 ; iniopes++ )
 for ( iscnt = 0 ; iscnt < 7 ; iscnt++ )
  for ( istrp = 0 ; istrp < 6 ; istrp++ )
   {
    sprintf( fn , "%s/lpop-io%d-sc%d-str%d" , fnbf , iniopes , iscnt , istrp ) ;
    b_t() ; /* start running internal clock */
    wr_lstr_orcl( fn , com , ndays , ndddfld , nddfld , ni , nj , nk , strp[ istrp ] , scnt[ iscnt ] , niopes[ iniopes ] , dbf , dbf_ ) ;
    rt = e_t( 0 ) ;
    if ( ip == 0 )
     printf( "case: T[ %f ] ISTRP[ %d ] SCNT[ %d ] IOPEs[ %d ]\n" , rt , strp[ istrp ] , ( int ) scnt[ iscnt ] , niopes[ iniopes ] ) ;
```



POP

Q2

4800 PEs, Q2	Time(s)	INS	FP_OP
Barotropic	220.285649	3362619394734242	10914798749862
Baroclinic	84.623336	638046552543018	123489441332158
T_avg	554.416994	10459543609613288	22070416032
Movie	98.516514	1838543581529579	15638400
TOTALs	957.842493	1.629875313842013e+16	134,426,326,136,452

Q4,e

4800 PEs, Q4	Time(s)	INS	FP_OP
Barotropic	162.845484	2493523139608176	10918903717734
Baroclinic	81.234007	611926226154622	123489442062604
T_avg	72.995206	1369947333186195	22070417409
Movie	12.397561	228560389936546	15640101
TOTALs	329.472258	4,703,957,088,885,539	134,430,431,837,848

Q4,s

9600 PEs, Q4	Time(s)	INS	FP_OP
Barotropic	143.867992	4352776136294947	11696471278395
Baroclinic	47.994133	755616085382567	133265275114487
T_avg	84.648207	3180959264572214	24868719153
Movie	13.812455	505002308418671	31278501
TOTALs	290.322787	8,794,353,794,668,399	144,986,646,390,536

Efficiency:

```
PES     : 1
TIME    : 0.343973315454068 (329472258 / 957842493)
INS     : 0.288608401448646 (4703957088885539 / 1.629875313842013e+16)
FP_OP   : 1.000030542390869 ( 134430431837848 /     134426326136452  )
```

Strong Scaling:

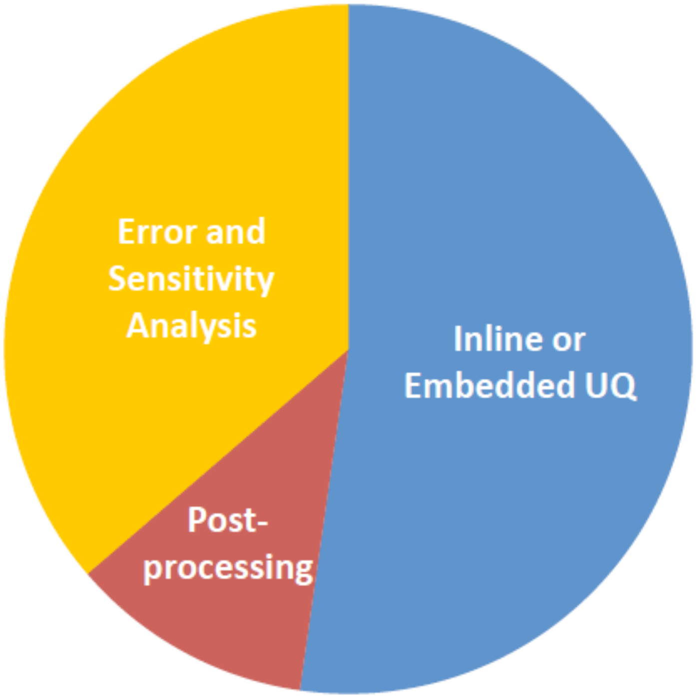
```
PES     : 2
TIME    : 0.3031007593855 (290.322787 / 957.842493)
INS     : 0.539572181993355 (8794353794668399 / 1.629875313842013e+16)
FP_OP   : 1.078558423469556 (144986646390536 / 134426326136452)
```

Computer Science in DOE

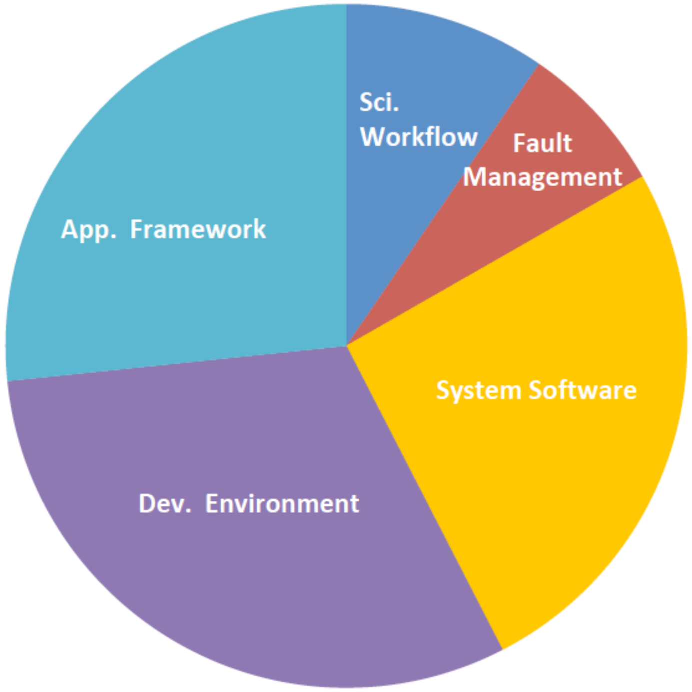
- advanced computer architectures
- programming models, languages, and compilers
- execution models, operating, runtime, and file systems
- performance and productivity tools
- data management and data analytics, visual analysis

any surprises / omissions?

ASCR Exascale Funding Trends

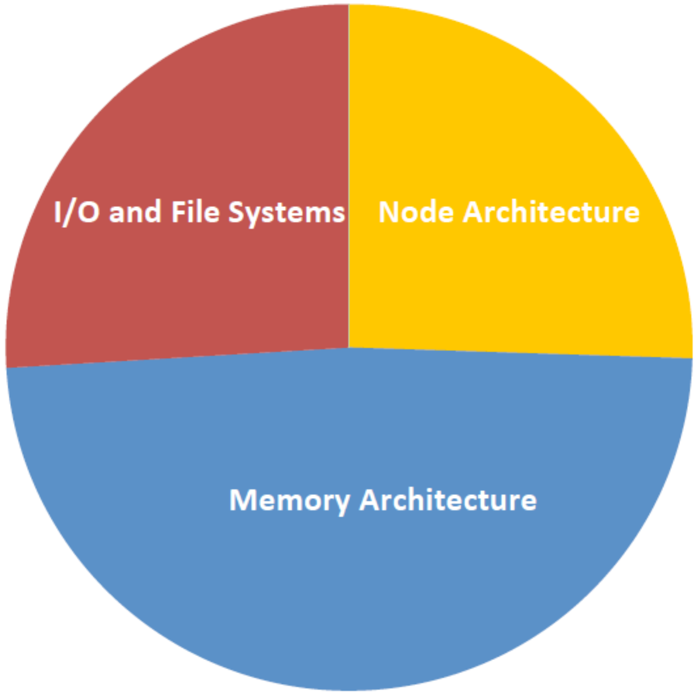


Uncertainty Quantification
6 funded at $3M/yr

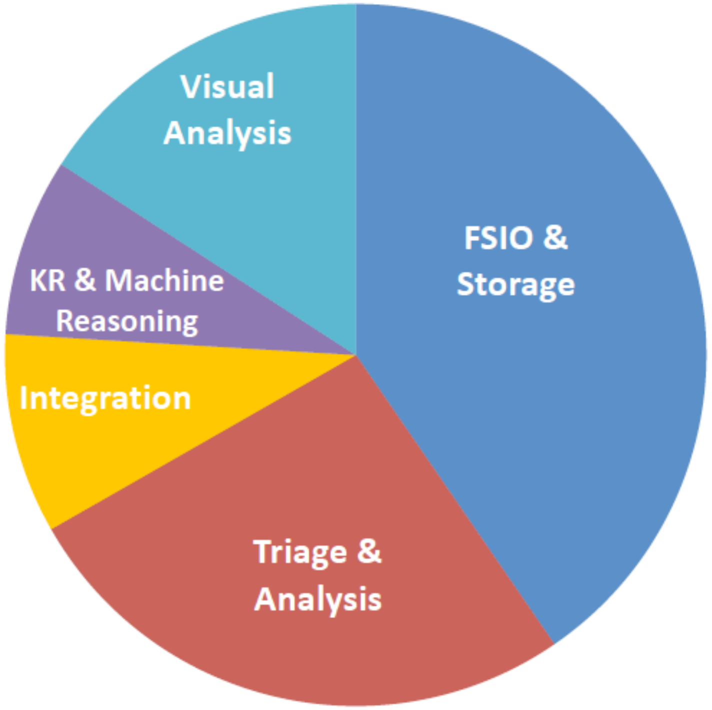


X-Stack
11 funded at $8.5M/yr

Advanced Architectures
6 funded at $5M/yr



Scientific Data Management and Analysis at Extreme Scale
10 (11?) projects funded at $5M/yr



advanced computer architectures

- the energy costs of moving data both on-chip and off-chip
- keeping the current technology roadmaps, memory per processor is expected to fall dramatically
- locality of data and computation renders flat cache hierarchies not useful
- energy-efficient on-chip and off-chip communication fabrics and synchronization mechanisms. Chief among these concerns is the power consumed by memory technology

programming models, languages, and compilers

- program up to a billion heterogeneous cores systems

- novel architectures /10 billion-way concurrency

- concurrency and locality

- includes development environments, frameworks, and debugging tools

- programming languages and environments

Performance is Limited by …

1) **System power** -primary constraint

2) **Memory** bandwidth and capacity are not keeping pace

3) **Concurrency** 1000X increase in-node

4) **Processor** open question

5) **Programming model** compilers will not hide this

6) **Algorithms** need to minimize data movement, not flops

7) **I/O bandwidth** unlikely to keep pace with machine speed

8) **Reliability and resiliency** will be critical at this scale

9) **Bisection bandwidth** limited by cost and energy

Bottom Line Challenges of Exascale Computing

Power efficiency,
Reliability,
Programmability