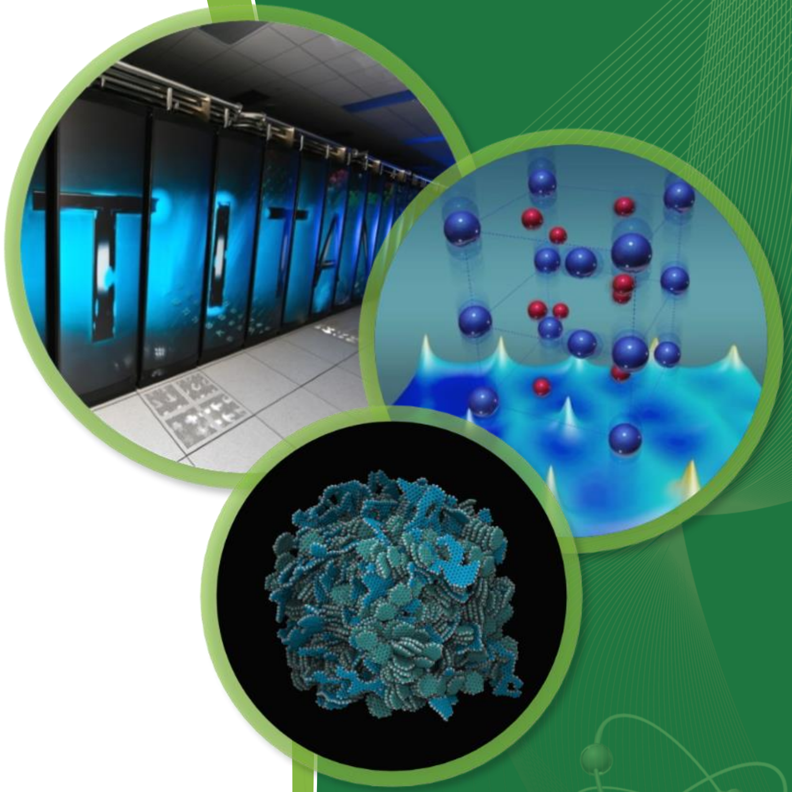
Using Application-Specific Performance Models to Inform Dynamic Scheduling

Jeffrey S. Vetter

Seyong Lee, Jeremy Meredith,

and many collaborators



Presented to
International Workshop on Runtime and Operating Systems for Supercomputers ROSS 2016

1 June 2016
Kyoto

Future Technologies Group

http://ft.ornl.gov vetter@computer.org

Executive Summary

- Both architectures and applications are growing more complex
  - Trends dictate that this will get worse, not better
  - This complexity creates irregularity in computation, communication, and data movement
- Dynamic resource management is one way to help manage this irregularity
  - Need accurate policies to guide resource decisions
  - Examples: greedy work stealing, algorithmic, historical, cost models, application specific, etc
- Posit that we can use application-specific performance models to inform scheduling decisions
  - Aspen performance modeling language helps create models
  - Two recent experiments
    - GPU offload
    - *Distributed scientific workflows*

Trends toward Exascale

Exascale architecture targets circa 2009
2009 Exascale Challenges Workshop in San Diego

Attendees envisioned two possible architectural swim lanes:
1. Homogeneous many-core thin-node system
2. Heterogeneous (accelerator + CPU) fat-node system

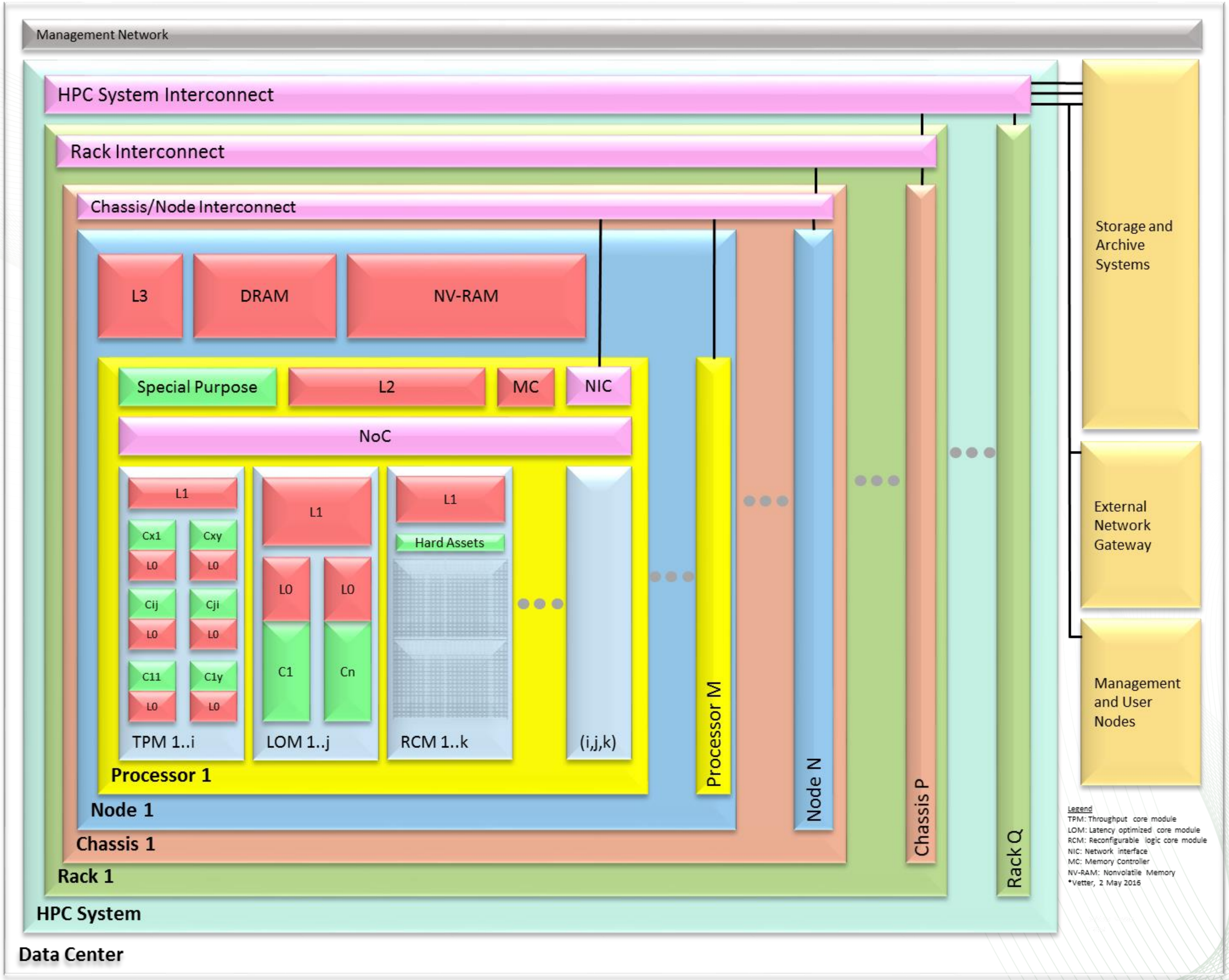
System attributes	2009	"Pre-Exascale"		"Exascale"	
System peak	2 PF	100-200 PF/s		1 Exaflop/s	
Power	6 MW	15 MW		20 MW	
System memory	0.3 PB	5 PB		32–64 PB	
Storage	15 PB	150 PB		500 PB	
Node performance	125 GF	0.5 TF	7 TF	1 TF	10 TF
Node memory BW	25 GB/s	0.1 TB/s	1 TB/s	0.4 TB/s	4 TB/s
Node concurrency	12	O(100)	O(1,000)	O(1,000)	O(10,000)
System size (nodes)	18,700	500,000	50,000	1,000,000	100,000
Node interconnect BW	1.5 GB/s	150 GB/s	1 TB/s	250 GB/s	2 TB/s
IO Bandwidth	0.2 TB/s	10 TB/s		30-60 TB/s	
MTTI	day	O(1 day)		O(0.1 day)	

Contemporary ASCR Computing At a Glance

System attributes	NERSC Now	OLCF Now	ALCF Now	NERSC Upgrade	OLCF Upgrade	ALCF Upgrades	
Planned Installation	**Edison**	**TITAN**	**MIRA**	**Cori 2016**	**Summit 2017-2018**	**Theta 2016**	**Aurora 2018-2019**
System peak (PF)	2.6	27	10	> 30	150	>8.5	180
Peak Power (MW)	2	9	4.8	< 3.7	10	1.7	13
Total system memory	357 TB	710TB	768TB	~1 PB DDR4 + High Bandwidth Memory (HBM)+1.5PB persistent memory	> 1.74 PB DDR4 + HBM + 2.8 PB persistent memory	>480 TB DDR4 + High Bandwidth Memory (HBM)	> 7 PB High Bandwidth On-Package Memory Local Memory and Persistent Memory
Node performance (TF)	0.460	1.452	0.204	> 3	> 40	> 3	> 17 times Mira
Node processors	Intel Ivy Bridge	AMD Opteron Nvidia Kepler	64-bit PowerPC A2	Intel Knights Landing many core CPUs Intel Haswell CPU in data partition	Multiple IBM Power9 CPUs & multiple Nvidia Voltas GPUS	Intel Knights Landing Xeon Phi many core CPUs	Knights Hill Xeon Phi many core CPUs
System size (nodes)	5,600 nodes	18,688 nodes	49,152	9,300 nodes 1,900 nodes in data partition	~3,500 nodes	>2,500 nodes	>50,000 nodes
System Interconnect	Aries	Gemini	5D Torus	Aries	Dual Rail EDR-IB	Aries	2nd Generation Intel Omni-Path Architecture
File System	7.6 PB 168 GB/s, Lustre®	32 PB 1 TB/s, Lustre®	26 PB 300 GB/s GPFS™	28 PB 744 GB/s Lustre®	120 PB 1 TB/s GPFS™	10PB, 210 GB/s Lustre initial	150 PB 1 TB/s Lustre®

Complexity α T

OAK RIDGE National Laboratory



Management Network
HPC System Interconnect
Rack Interconnect
Chassis/Node Interconnect
L3
DRAM
NV-RAM
Special Purpose
L2
MC
NIC
NoC
L1
Cx1
Cxy
L0
L0
Cij
Cji
L0
L0
C11
C1y
L0
L0
TPM 1..i
L1
L0
L0
C1
Cn
LOM 1..j
L1
Hard Assets
RCM 1..k
(i,j,k)
Processor 1
Processor M
Node 1
Node N
Chassis 1
Chassis P
Rack 1
Rack Q
HPC System
Data Center
Storage and Archive Systems
External Network Gateway
Management and User Nodes
Legend
TPM: Throughput core module
LOM: Latency optimized core module
RCM: Reconfigurable logic core module
NIC: Network interface
MC: Memory Controller
NV-RAM: Nonvolatile Memory
*Vetter, 2 May 2016

Complexity is the next major challenge!

- "Exciting" times in computer architecture
  - Heterogeneous cores
  - Multimode memory systems
  - Fused memory systems
  - I/O architectures
  - Error correction
  - Changing system balance
- Uncertainty, Ambiguity
  - How do we design future systems so that they are faster than current systems on mission applications?
    - Entirely possible that the new system will be slower than the old system!
  - How do we provide some level of performance portability for applications teams?
  - How do we understand reliability and performance problems?
- Managing complexity is our main challenge!

Performance Prediction with Aspen

Example Ad Hoc Model: Latex Equations

the communication and computation of two sheets. The expression we get for the runtime is

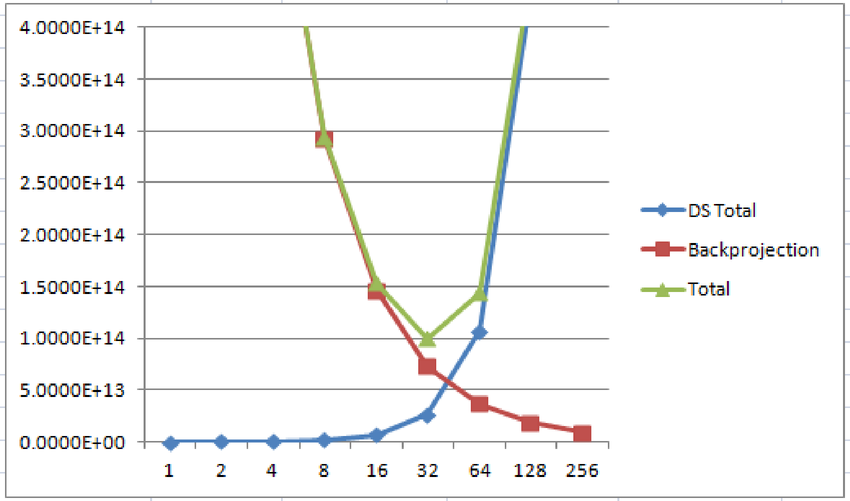
$$T = 2\left[t_c \frac{n}{p} n \log_2 n + (p-1)o + (p-2)g + \frac{n}{p} nG \right.$$
$$+ L + \left(\frac{n}{p} - 1\right) \max\left\{(p-1)o + t_c \frac{n}{p} n \log_2 n,\right.$$
$$\left.\left.(p-1)g + \frac{n}{p} nG + L\right\}\right] + t_c \frac{n^2}{p^2} n \log_2 n$$

Example: Ad-Hoc Excel Files

DS = Digital Spotlighting

Tile Factor	DS Pulses/Sec	DS Samples/Pulse	DS FFT Flops	DS Range/Pulse	DS Total	Backprojection		Total
1	2809	80636	1.85E+10	1.0442E+11	1.2288E+11	2.3378E+15		2.3380E+15
2	1405	40318	1.85E+10	3.1889E+10	2.0140E+11	1.1693E+15		1.1695E+15
4	703	20159	1.85E+10	1.3753E+10	5.1539E+11	5.8509E+14		5.8560E+14
8	352	10080	1.85E+10	9.2163E+09	1.7712E+12	2.9296E+14		2.9473E+14
16	176	5040	1.85E+10	8.0800E+09	6.7941E+12	1.4648E+14		1.5327E+14
32	88	2520	1.85E+10	7.7960E+09	2.6885E+13	7.3240E+13		1.0013E+14
64	44	1260	1.85E+10	7.7249E+09	1.0725E+14	3.6620E+13		1.4387E+14
128	22	630	1.85E+10	7.7072E+09	4.2871E+14	1.8310E+13		4.4702E+14
256	11	315	1.85E+10	7.7027E+09	1.7146E+15	9.1550E+12		1.7237E+15

* Note: The DS FFT flops category is missing the initial FFT in range for each pulse. However, this only needs to be done once at a cost of ~2e10 flop



Prediction Techniques Ranked

	Speed	Ease	Flexibility	Accuracy	Scalability
Ad-hoc Analytical Models	1	3	2	4	1
Structured Analytical Models	1	2	1	4	1
Aspen	1	1	1	4	1
Simulation – Functional	3	2	2	3	3
Simulation – Cycle Accurate	4	2	2	2	4
Hardware Emulation (FPGA)	3	3	3	2	3
Similar hardware measurement	2	1	4	2	2
Node Prototype	2	1	4	1	4
Prototype at Scale	2	1	4	1	2
Final System	-	-	-	-	-

Aspen: Abstract Scalable Performance Engineering Notation

Model Creation

- Static analysis via compiler, tools
- Empirical, Historical
- Manual (for future applications)

Representation in Aspen
- **Modular**
- **Sharable**
- **Composable**
- **Reflects prog structure**

Model Uses

- Interactive tools for graphs, queries
- Design space exploration
- Workload Generation
- Feedback to Runtime Systems

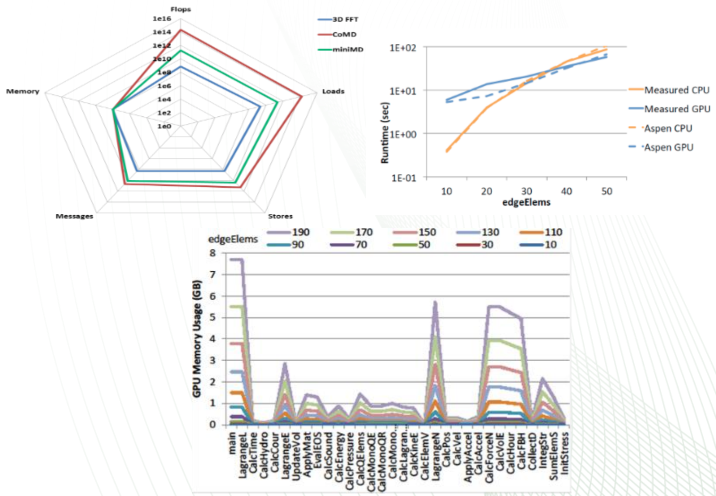
E.g., MD, UHPC CP 1, Lulesh, 3D FFT, CoMD, VPFFT, …

Source code

```
static inline
void CalcMonotonicQGradientsForElems(Index_t p_nodelist[T_NUMELEM8],
        Real_t p_x[T_NUMNODE], Real_t p_y[T_NUMNODE], Real_t p_z[T_NUMNODE],
        Real_t p_xd[T_NUMNODE], Real_t p_yd[T_NUMNODE],Real_t p_zd[T_NUMNODE],
        Real_t p_volo[T_NUMELEM], Real_t p_vnew[T_NUMELEM],
        Real_t p_delx_zeta[T_NUMELEM], Real_t p_delv_zeta[T_NUMELEM],
        Real_t p_delx_xi[T_NUMELEM], Real_t p_delv_xi[T_NUMELEM],
        Real_t p_delx_eta[T_NUMELEM], Real_t p_delv_eta[T_NUMELEM])
{
   Index_t i;
   Index_t numElem = m_numElem;
#pragma acc parallel loop independent present(p_vnew, p_nodelist, p_x, p_y, p_z, p_xd, \
p_yd, p_zd, p_volo, p_delx_xi, p_delx_eta, p_delx_zeta, p_delv_xi, p_delv_eta,\
p_delv_zeta)
   for (i = 0 ; i < numElem ; ++i ) {
      const Real_t ptiny = 1.e-36 ;
      Real_t ax,ay,az ;
      Real_t dxv,dyv,dzv ;

      const Index_t *elemToNode = &p_nodelist[8*i];
      Index_t n0 = elemToNode[0] ;
      Index_t n1 = elemToNode[1] ;
      Index_t n2 = elemToNode[2] ;
      Index_t n3 = elemToNode[3] ;
      Index_t n4 = elemToNode[4] ;
      Index_t n5 = elemToNode[5] ;
      Index_t n6 = elemToNode[6] ;
      Index_t n7 = elemToNode[7] ;

      Real_t x0 = p_x[n0] ;
```

Aspen code

```
kernel CalcMonotonicQGradients {
  execute [numElems]
  {
    loads [8 * indexWordSize] from nodelist
    // Load and cache position and velocity.
    loads/caching [8 * wordSize] from x
    loads/caching [8 * wordSize] from y
    loads/caching [8 * wordSize] from z

    loads/caching [8 * wordSize] from xvel
    loads/caching [8 * wordSize] from yvel
    loads/caching [8 * wordSize] from zvel

    loads [wordSize] from volo
    loads [wordSize] from vnew
    // dx, dy, etc.
    flops [90] as dp, simd
    // delvk delxk
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_xeta
    // delxi delvi
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delx_xi
    // delxj and delvj
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_eta
  }
}
```



K. Spafford and J.S. Vetter, "Aspen: A Domain Specific Language for Performance Modeling," in Proc. *SC12*.

Creating an Aspen Model

Manual Example of LULESH

branch: master | aspen / models / lulesh / **lulesh.aspen**

jsmeredith on Sep 20, 2013 adding models

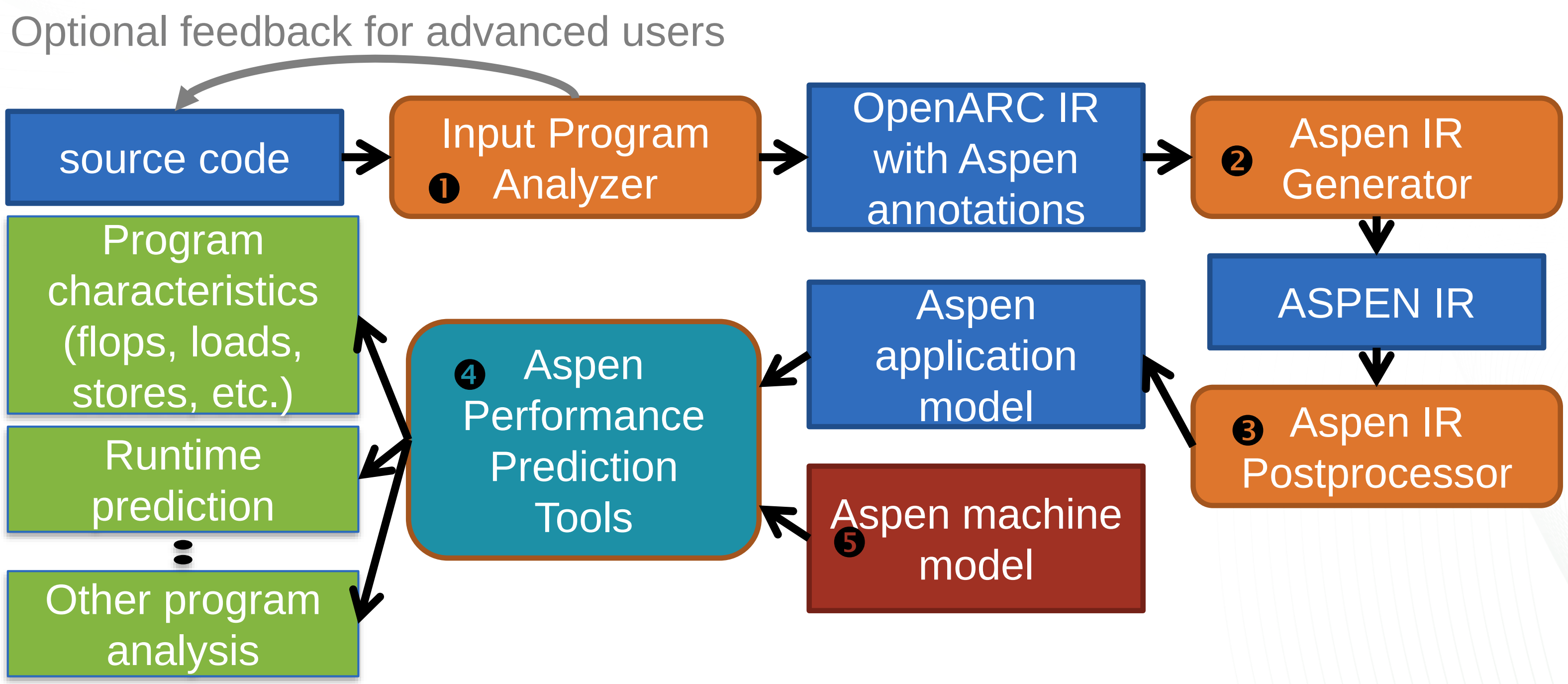
1 contributor

336 lines (288 sloc) | 9.213 kb   Raw | Blame | History

```
//
// lulesh.aspen
//
// An ASPEN application model for the LULESH 1.01 challenge problem.  Based
// on the CUDA version of the source code found at:
//  https://computation.llnl.gov/casc/ShockHydro/
//
param nTimeSteps = 1495

// Information about domain
param edgeElems = 45
param edgeNodes = edgeElems + 1

param numElems = edgeElems^3
param numNodes = edgeNodes^3

// Double precision
param wordSize = 8

// Element data
data mNodeList as Array(numElems, wordSize)
data mMatElemList as Array(numElems, wordSize)
data mNodeList as Array(8 * numElems, wordSize) //  8 nodes per element
data mlxim as Array(numElems, wordSize)
data mlxip as Array(numElems, wordSize)
data mletam as Array(numElems, wordSize)
data mletap as Array(numElems, wordSize)
data mzetam as Array(numElems, wordSize)
data mzetap as Array(numElems, wordSize)
data melemBC as Array(numElems, wordSize)
data mE as Array(numElems, wordSize)
data mP as Array(numElems, wordSize)
```

```
kernel CalcMonotonicQGradients {
  execute [numElems]
  {
    loads [8 * indexWordSize] from nodelist
    // Load and cache position and velocity.
    loads/caching [8 * wordSize] from x
    loads/caching [8 * wordSize] from y
    loads/caching [8 * wordSize] from z

    loads/caching [8 * wordSize] from xvel
    loads/caching [8 * wordSize] from yvel
    loads/caching [8 * wordSize] from zvel

    loads [wordSize] from volo
    loads [wordSize] from vnew
    // dx, dy, etc.
    flops [90] as dp, simd
    // delvk delxk
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_xeta
    // delxi delvi
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delx_xi
    // delxj and delvj
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_eta
  }
}
```

COMPASS System Overview



S. Lee, J.S. Meredith, and J.S. Vetter, "COMPASS: A Framework for Automated Performance Modeling and Prediction," in *ACM International Conference on Supercomputing (ICS). Newport Beach, California: ACM, 2015, 10.1145/2751205.2751220.*

OAK RIDGE National Laboratory

MM example generated from COMPASS

```
int N = 1024;
void matmul(float *a, float *b, float *c){ int i, j, k ;
#pragma acc kernels loop gang copyout(a[0:(N*N)]) \
copyin(b[0:(N*N)],c[0:(N*N)])
  for (i=0; i<N; i++){
#pragma acc loop worker
    for (j=0; j<N; j++) { float sum = 0.0 ;
      for (k=0; k<N; k++) {sum+=b[i*N+k]*c[k*N+j];}
      a[i*N+j] = sum; }
  } //end of i loop
} //end of matmul()
int main() {
  int i; float *A = (float*) malloc(N*N*sizeof(float));
  float *B = (float*) malloc(N*N*sizeof(float));
  float *C = (float*) malloc(N*N*sizeof(float));
  for (i = 0; i <  N*N; i++)
  {  A[i] = 0.0F; B[i] = (float) i; C[i] = 1.0F; }
#pragma aspen modelregion label(MM)
  matmul(A,B,C);
  free(A); free(B); free(C); return 0;
} //end of main()
```

```
model MM {
  param floatS = 4; param N = 1024
  data A as Array((N*N), floatS)
  data B as Array((N*N), floatS)
  data C as Array((N*N), floatS)
  kernel matmul {
    execute matmul2_intracommIN
    {  intracomm [floatS*(N*N)] to C as copyin
      intracomm [floatS*(N*N)] to B as copyin }
    map matmul2 [N] {
      map matmul3 [N] {
        iterate [N] {
          execute matmul5
          {  loads [floatS] from B as stride(1)
            loads [floatS] from C; flops [2] as sp, simd }
        } //end of iterate
        execute matmul6 { stores [floatS] to A as stride(1) }
      } // end of map matmul3
    } //end of map matmul2
    execute matmul2_intracommOUT
    { intracomm [floatS*(N*N)] to A as copyout }
  } //end of kernel matmul
  kernel main { matmul() }
} //end of model MM
```

Example: LULESH (10% of 1 kernel)

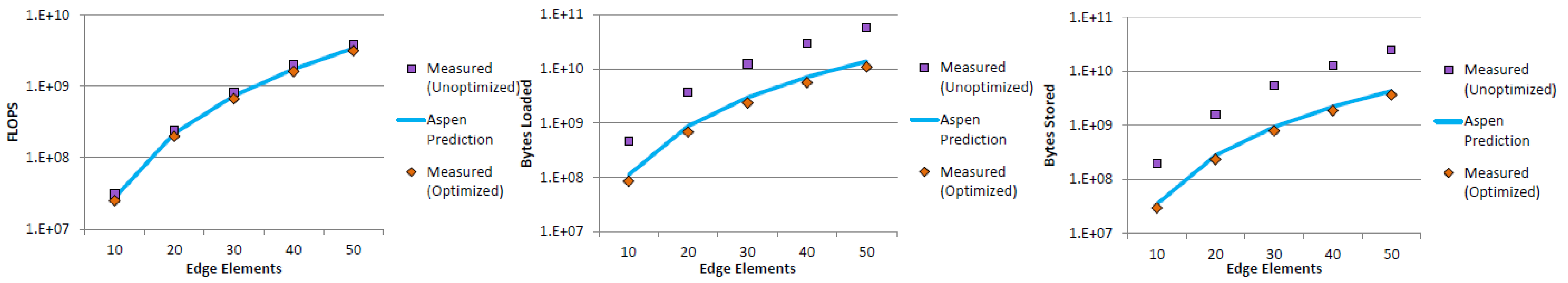
```
kernel IntegrateStressForElems
{
  execute [numElem_CalcVolumeForceForElems]
  {
     loads [((1*aspen_param_int)*8)] from elemNodes as stride(1)
     loads [((1*aspen_param_double)*8)] from m_x
     loads [((1*aspen_param_double)*8)] from m_y
     loads [((1*aspen_param_double)*8)] from m_z
     loads [(1*aspen_param_double)] from determ as stride(1)
     flops [8] as dp, simd
     flops [8] as dp, simd
     flops [8] as dp, simd
     flops [8] as dp, simd
     flops [3] as dp, simd
     flops [3] as dp, simd
     flops [3] as dp, simd
     flops [3] as dp, simd
     stores [(1*aspen_param_double)] as stride(0)
     flops [2] as dp, simd
     stores [(1*aspen_param_double)] as stride(0)
     flops [2] as dp, simd
     stores [(1*aspen_param_double)] as stride(0)
     flops [2] as dp, simd
     loads [(1*aspen_param_double)] as stride(0)
     stores [(1*aspen_param_double)] as stride(0)
     loads [(1*aspen_param_double)] as stride(0)
     stores [(1*aspen_param_double)] as stride(0)
     loads [(1*aspen_param_double)] as stride(0)
     . . . . . .
```

- Input LULESH program: 3700 lines of C codes
- Output Aspen model: 2300 lines of Aspen codes

Model Validation

	FLOPS	LOADS	STORES
MATMUL	15%	<1%	1%
LAPLACE2D	7%	0%	<1%
SRAD	17%	0%	0%
JACOBI	6%	<1%	<1%
KMEANS	0%	0%	8%
LUD	5%	0%	2%
BFS	<1%	11%	0%
HOTSPOT	0%	0%	0%
LULESH	0%	0%	0%

0% means that prediction fell between measurements from optimized and unoptimized runs of the code.

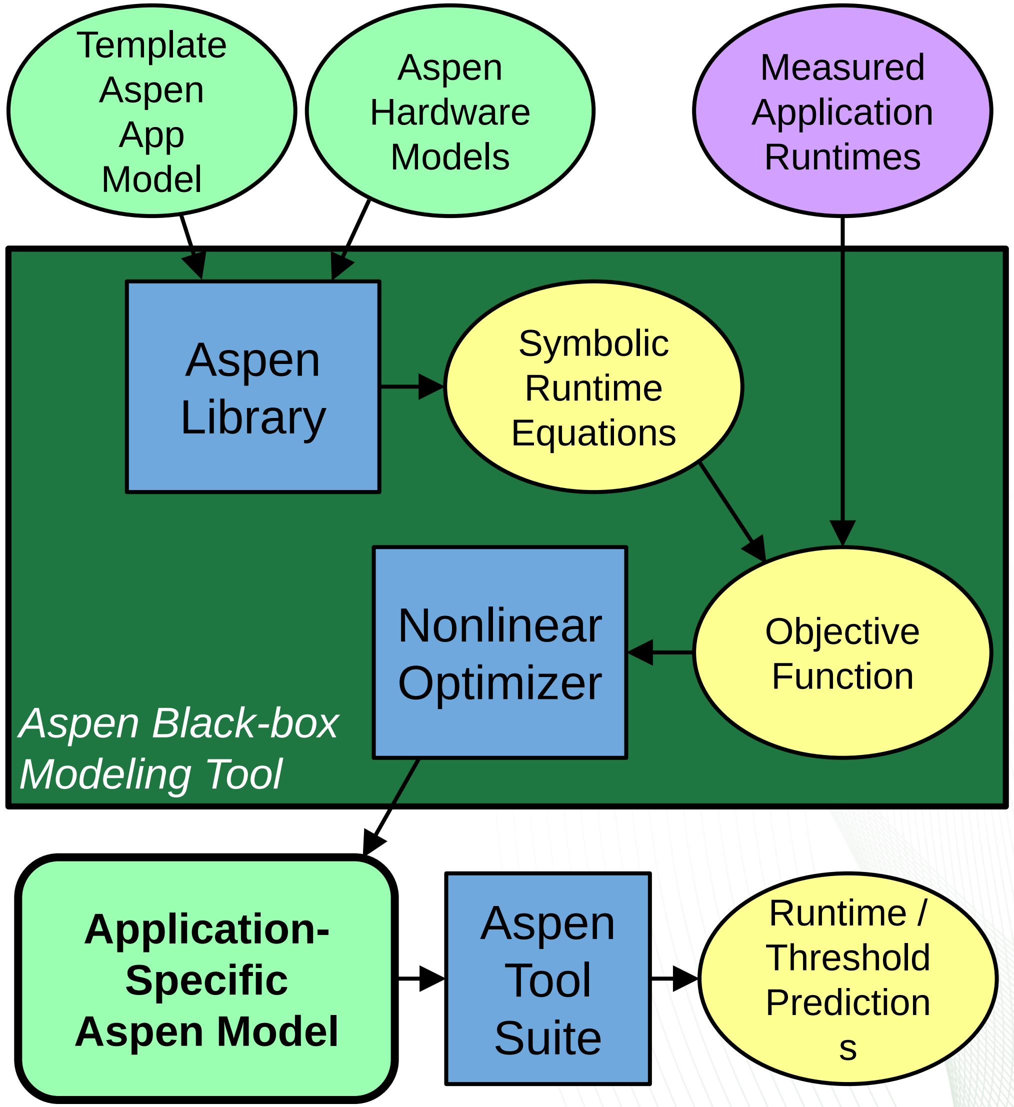


(a) Floating Point Operations (b) Loads in Bytes (c) Stores in Bytes

Figure 2: Predicted Resource Usage of LULESH versus Measured (with and without compiler optimization)

Black Box Analytical Modeling

- In some cases, we do not have access to a white box Aspen performance model
- Using input vector and empirical results, we can develop Aspen Black Box model
- User provides
  - measured runtimes with app/machine parameters
    - e.g. nAtoms, nCores
  - template Aspen model with
    - application parameters
    - unknowns to solve for
  - new machine models (if necessary)
- Modeling tool
  - Generates symbolic predictions
  - Combines with measurements to generate objective function
  - Solves for unknowns in template
  - Output: completed app model usable for predictive behavior



OAK RIDGE National Laboratory

Black Box Modeling Example

MD template model

```
model NAMD_Template {
  // application parameters
  // (defined in the input file)
  param nAtoms     = 1e6
  param nTimeSteps = 100
  // solve for these parameters
  // (within the given ranges)
  param c = 1 in 1 .. 1e18
  param d = 1 in 1 .. 1e18
  // application behavior:
  // execution and control flow
  kernel main
  {
    iterate [nTimeSteps] {
      execute {
        loads [c * nAtoms^2]
        flops [d * nAtoms]
      }
    }
  }
}
```

+

CSV data file with parameters and runtimes

nAtoms	nTimeSteps	nCores	machine	runtime
1e6	100	144	exogeni	384.2
1e6	100	144	hopper	340.1
1e6	150	144	hopper	482.9

=

Concrete NAMD model

```
model NAMD_Equilibrate {

  // NAMD input parameters
  param nAtoms     = 1e6
  param nTimeSteps = 100

  // calculation-specific constants
  param c = 402.1
  param d = 10.95

  // NAMD application behavior
  kernel main
  {
    iterate [nTimeSteps] {
      execute {
        loads [c * nAtoms^2]
        flops [d * nAtoms]
      }
    }
  }
}
```

- nAtoms and nTimeSteps defined in template application model and CSV input data
- nCores defined in machine models and CSV input data
- solves for c and d, filling out a concrete application model for that problem
- new predictions can still vary nAtoms, nTimeSteps, and nCores

Using an Aspen Model

Aspen: Abstract Scalable Performance Engineering Notation

Model Creation

- Static analysis via compiler, tools
- Empirical, Historical
- Manual (for future applications)

Representation in Aspen

- **Modular**
- **Sharable**
- **Composable**
- **Reflects prog structure**

Model Uses

- Interactive tools for graphs, queries
- Design space exploration
- Workload Generation
- Feedback to Runtime Systems

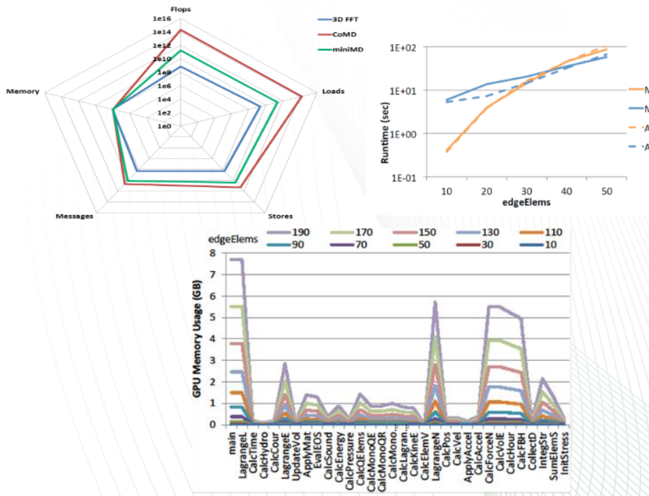
E.g., MD, UHPC CP 1, Lulesh, 3D FFT, CoMD, VPFFT, …

Source code

```
static inline
void CalcMonotonicQGradientsForElems(Index_t p_nodelist[T_NUMELEM8],
        Real_t p_x[T_NUMNODE], Real_t p_y[T_NUMNODE], Real_t p_z[T_NUMNODE],
        Real_t p_xd[T_NUMNODE], Real_t p_yd[T_NUMNODE],Real_t p_zd[T_NUMNODE],
        Real_t p_volo[T_NUMELEM], Real_t p_vnew[T_NUMELEM],
        Real_t p_delx_zeta[T_NUMELEM], Real_t p_delv_zeta[T_NUMELEM],
        Real_t p_delx_xi[T_NUMELEM], Real_t p_delv_xi[T_NUMELEM],
        Real_t p_delx_eta[T_NUMELEM], Real_t p_delv_eta[T_NUMELEM])
{
   Index_t i;
   Index_t numElem = m_numElem;
#pragma acc parallel loop independent present(p_vnew, p_nodelist, p_x, p_y, p_z, p_xd, \
p_yd, p_zd, p_volo, p_delx_xi, p_delx_eta, p_delx_zeta, p_delv_xi, p_delv_eta,\
p_delv_zeta)
   for (i = 0 ; i < numElem ; ++i ) {
      const Real_t ptiny = 1.e-36 ;
      Real_t ax,ay,az ;
      Real_t dxv,dyv,dzv ;

      const Index_t *elemToNode = &p_nodelist[8*i];
      Index_t n0 = elemToNode[0] ;
      Index_t n1 = elemToNode[1] ;
      Index_t n2 = elemToNode[2] ;
      Index_t n3 = elemToNode[3] ;
      Index_t n4 = elemToNode[4] ;
      Index_t n5 = elemToNode[5] ;
      Index_t n6 = elemToNode[6] ;
      Index_t n7 = elemToNode[7] ;

      Real_t x0 = p_x[n0] ;
```

Aspen code

```
kernel CalcMonotonicQGradients {
  execute [numElems]
  {
    loads [8 * indexWordSize] from nodelist
    // Load and cache position and velocity.
    loads/caching [8 * wordSize] from x
    loads/caching [8 * wordSize] from y
    loads/caching [8 * wordSize] from z

    loads/caching [8 * wordSize] from xvel
    loads/caching [8 * wordSize] from yvel
    loads/caching [8 * wordSize] from zvel

    loads [wordSize] from volo
    loads [wordSize] from vnew
    // dx, dy, etc.
    flops [90] as dp, simd
    // delvk delxk
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_xeta
    // delxi delvi
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delx_xi
    // delxj and delvj
    flops [9 + 8 + 3 + 30 + 5] as dp, simd
    stores [wordSize] to delv_eta
  }
}
```

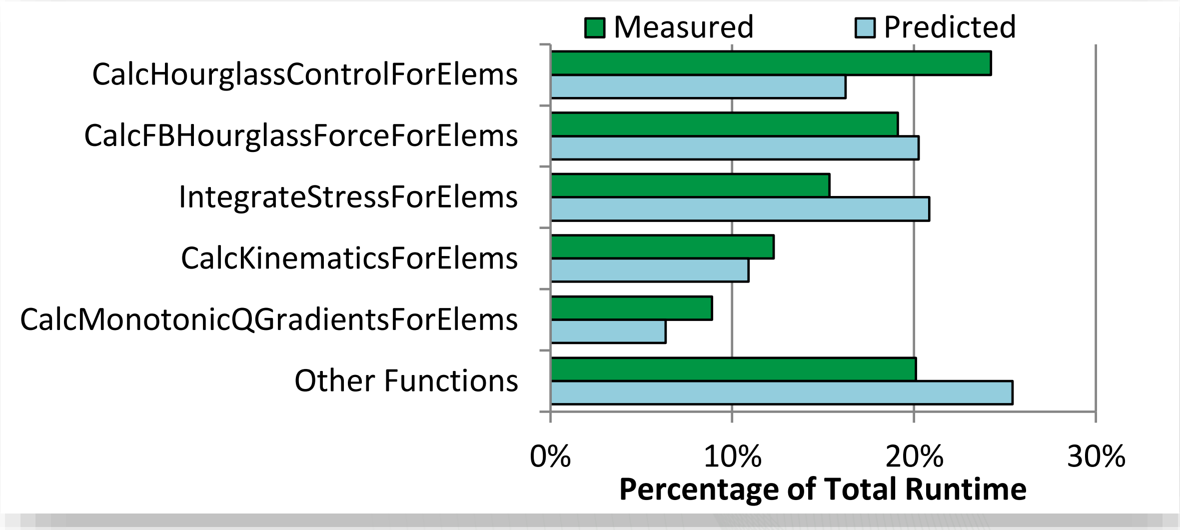


K. Spafford and J.S. Vetter, "Aspen: A Domain Specific Language for Performance Modeling," in Proc. *SC12*.

View Aspen performance models as normal performance analysis output with Gprof

```
Flat profile:

  %    cum     self          self      total
 time  sec     sec  calls ms/call    ms/call    name
--------------------------------------------------------------
86.91 370.76 370.76   30 12358.52   12358.52  fft3d.localFFT
10.03 413.54  42.78   20  2139.09    2139.09  fft3d.exchange
 3.06 426.57  13.03   20   651.71     651.71  fft3d.shuffle
 0.00 426.57   0.00   10     0.03       0.03  exchange
 0.00 426.57   0.00   10     0.03       0.03  buildNList
 0.00 426.57   0.00   10     0.01       0.01  ljForce
 0.00 426.57   0.00   30     0.00       0.00  integrate
 0.00 426.57   0.00   10     0.00   42657.18  fft
 0.00 426.57   0.00   10     0.00   42657.18  fft3d.main
 0.00 426.57   0.00    1     0.00  426572.70  main

Call graph:
index  %time      self  children        name
--------------------------------------------------------------
[  1]  100.0      0.00   426.57         main [1]
                  0.00     0.00             buildNList [8]
                  0.00     0.00             exchange [7]
                  0.00   426.57             fft [2]
                  0.00     0.00             integrate [10]
                  0.00     0.00             ljForce [9]
--------------------------------------------------------------
                  0.00   426.57             main [1]
[  2]  100.0      0.00   426.57         fft [2]
                  0.00   426.57             fft3d.main [3]
--------------------------------------------------------------
                  0.00   426.57             fft [2]
[  3]  100.0      0.00   426.57         fft3d.main [3]
                 42.78     0.00             fft3d.exchange [5]
                370.76     0.00             fft3d.localFFT [4]
                 13.03     0.00             fft3d.shuffle [6]
--------------------------------------------------------------
                  0.00   426.57             fft3d.main [3]
[  4]   86.9    370.76     0.00          fft3d.localFFT [4]
--------------------------------------------------------------
                  0.00   426.57             fft3d.main [3]
[  5]   10.0     42.78     0.00         fft3d.exchange [5]
--------------------------------------------------------------
                  0.00   426.57             fft3d.main [3]
[  6]    3.1     13.03     0.00         fft3d.shuffle [6]
--------------------------------------------------------------
                  0.00   426.57             main [1]
[  7]    0.0      0.00     0.00         exchange [7]
--------------------------------------------------------------
                  0.00   426.57             main [1]
[  8]    0.0      0.00     0.00         buildNList [8]
--------------------------------------------------------------
                  0.00   426.57             main [1]
[  9]    0.0      0.00     0.00         ljForce [9]
--------------------------------------------------------------
                  0.00   426.57             main [1]
[ 10]    0.0      0.00     0.00         integrate [10]
```

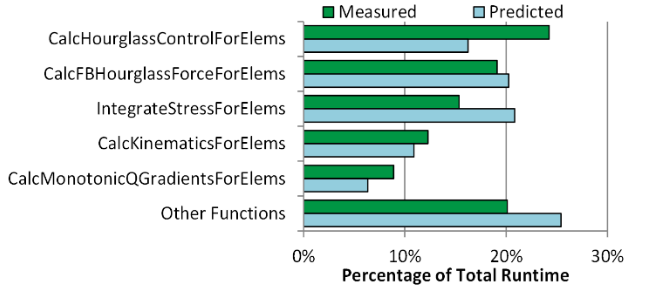
Not from gprof but rather aspen performance model



Aspen Model User Queries

Benchmark	Runtime Order
BACKPROP	$H * O + H * I$
BFS	$nodes + edges$
CFD	$nelr * ndim$
CG	$nrow + ncol$
HOTSPOT	$sim_time * rows * cols$
JACOBI	$m\_size * m\_size$
KMEANS	$nAttr * nClusters$
LAPLACE2D	n^2
LUD	$matrix\_dim^3$
MATMUL	$N * M * P$
NW	max_cols^2
SPMUL	$size + nonzero$
SRAD	$niter * rows * cols$

Table 2: Order analysis, showing Big O runtime for each benchmark in terms of its key parameters.



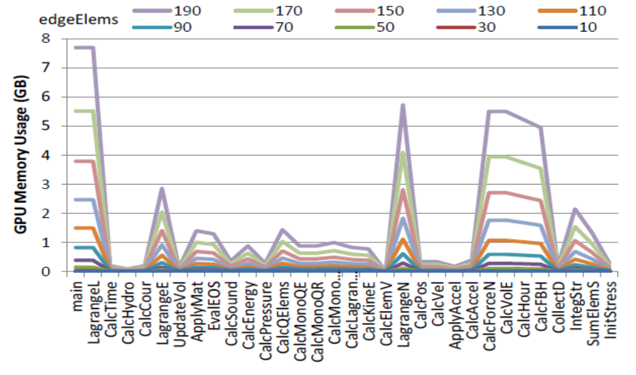


Fig. 8: GPU Memory Usage of each Function in LULESH, where the memory usage of a function is inclusive; value for a parent function includes data accessed by its child functions in the call graph.

Method Name	FLOPS/byte
InitStressTermsForElems	0.03
CalcElemShapeFunctionDerivatives	0.44
SumElemFaceNormal	0.50
CalcElemNodeNormals	0.15
SumElemStressesToNodeForces	0.06
IntegrateStressForElems	0.15
CollectDomainNodesToElemNodes	0.00
VoluDer	1.50
CalcElemVolumeDerivative	0.33
CalcElemFBHourglassForce	0.15
CalcFBHourglassForceForElems	0.17
CalcHourglassControlForElems	0.19
CalcVolumeForceForElems	0.18
CalcForceForNodes	0.18
CalcAccelerationForNodes	0.04
ApplyAccelerationBoundaryCond	0.00
CalcVelocityForNodes	0.13
CalcPositionForNodes	0.13
LagrangeNodal	0.18
AreaFace	10.25
CalcElemCharacteristicLength	0.44
CalcElemVelocityGrandient	0.13
CalcKinematicsForElems	0.24
CalcLagrangeElements	0.24
CalcMonotonicQGradientsForElems	0.46

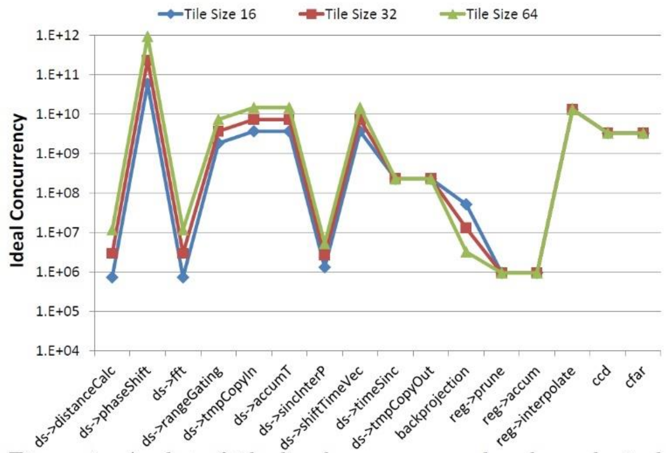


Figure 1: A plot of idealized concurrency by chronological phase in the digital spotlighting application model.

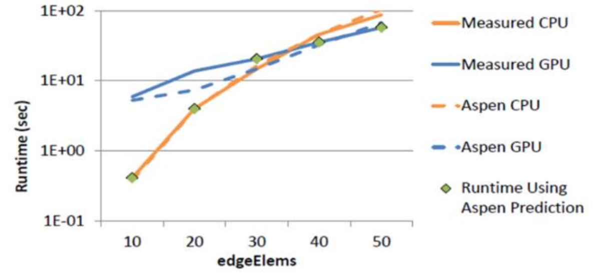


Fig. 7: Measured and predicted runtime of the entire LULESH program on CPU and GPU, including measured runtimes using the automatically predicted optimal target device at each size.

Scheduling GPU Offloads with Aspen Performance Models

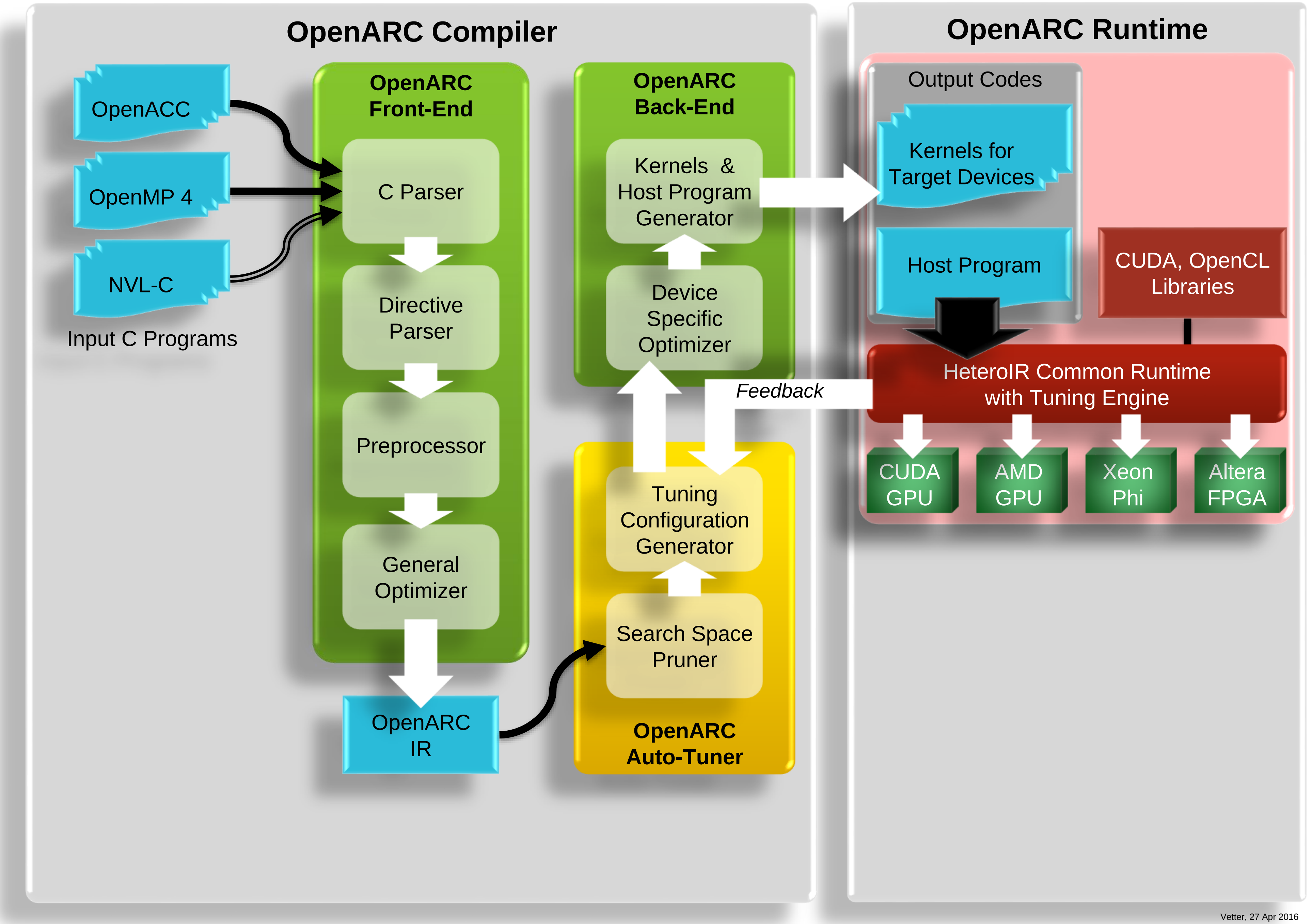
Should the execution offload kernel to GPU or run on host CPU?

- Intuitively
  - When it is ‘small’, run the computation on the host CPU,
  - Otherwise, send it to the GPU
  - Expense of data movement (twice) and launch GPU kernels?
    - Depends on architecture
  - Simply offloading all computation is not smart

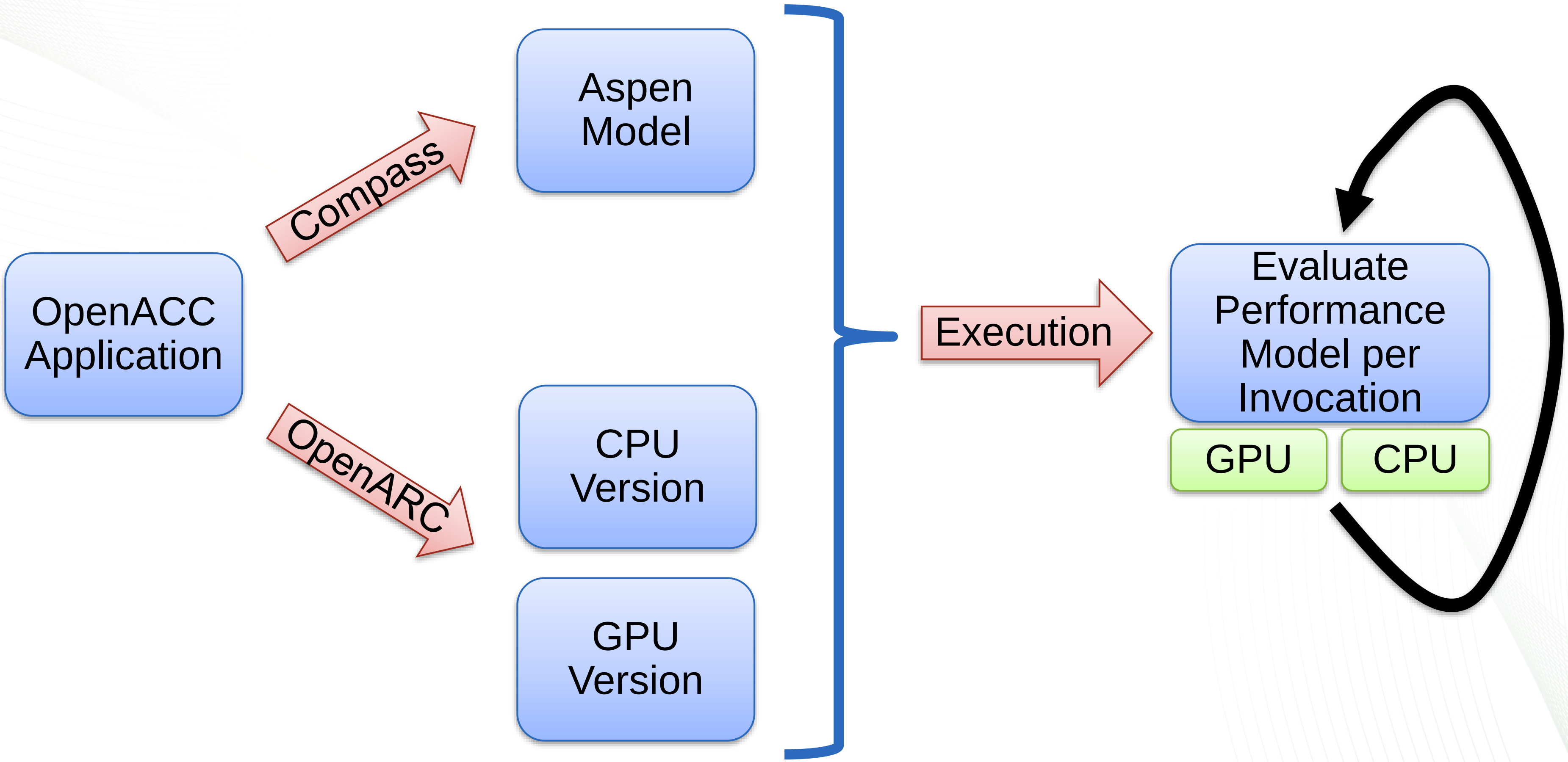
- Portability?
  - Need to account for performance, working set size, data transfer costs, …

Listing 1: Input OpenACC Matrix Multiplication Code

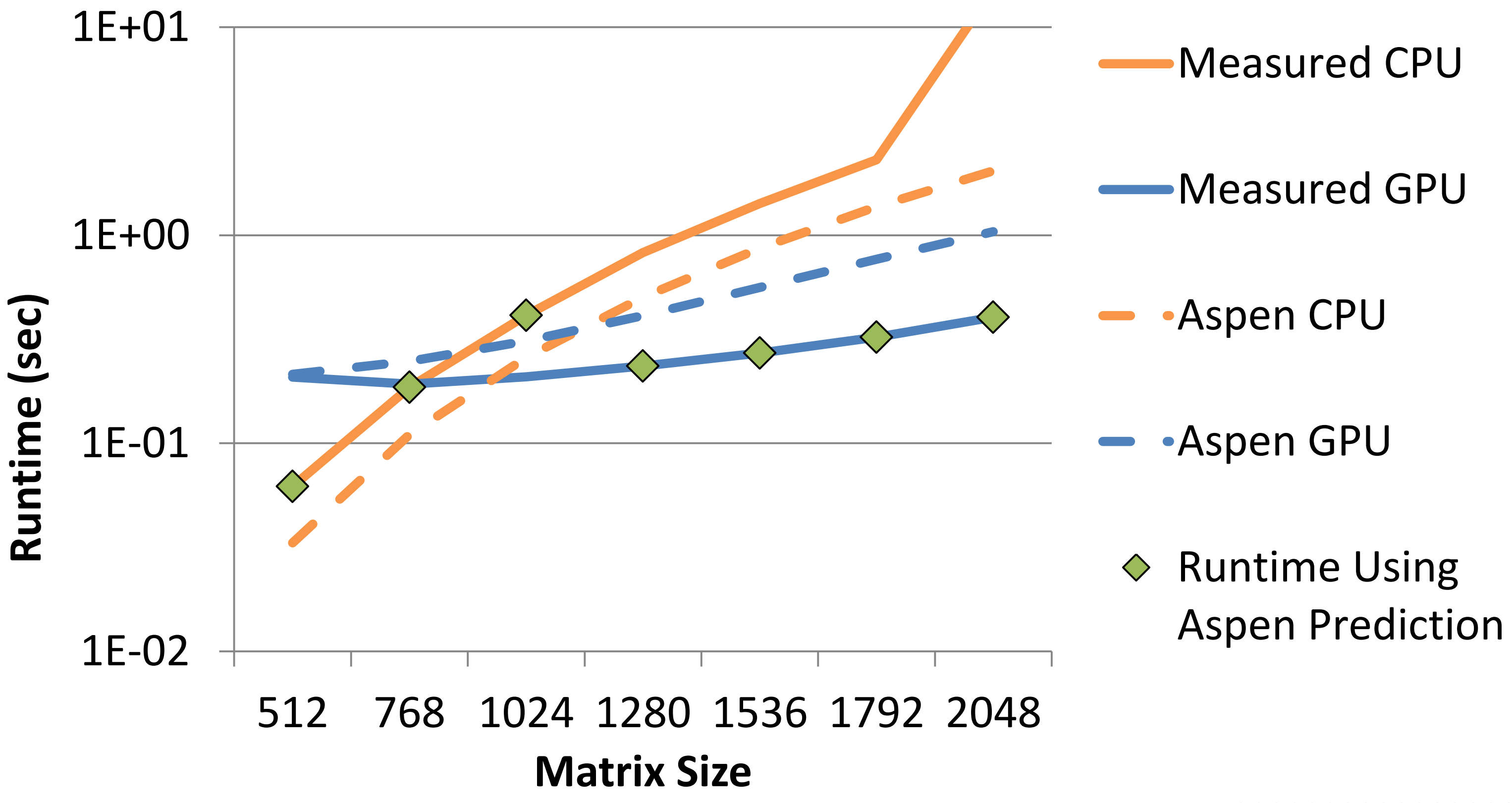
```
int N = 1024;
void matmul(float *a, float *b, float *c){ int i, j, k ;
#pragma acc kernels loop gang copyout(a[0:(N*N)]) \
copyin(b[0:(N*N)],c[0:(N*N)])
  for (i=0; i<N; i++){
#pragma acc loop worker
    for (j=0; j<N; j++) { float sum = 0.0 ;
      for (k=0; k<N; k++) {sum+=b[i*N+k]*c[k*N+j];}
      a[i*N+j] = sum; }
  } //end of i loop
} //end of matmul()
int main() {
  int i; float *A = (float*) malloc(N*N*sizeof(float));
  float *B = (float*) malloc(N*N*sizeof(float));
  float *C = (float*) malloc(N*N*sizeof(float));
  for (i = 0; i <  N*N; i++)
  {  A[i] = 0.0F; B[i] = (float) i; C[i] = 1.0F; }
#pragma aspen modelregion label(MM)
  matmul(A,B,C);
  free(A); free(B); free(C); return 0;
} //end of main()
```


OpenARC Compiler
OpenARC Runtime
OpenACC
OpenMP 4
NVL-C
Input C Programs
OpenARC
Front-End
C Parser
Directive
Parser
Preprocessor
General
Optimizer
OpenARC
IR
OpenARC
Back-End
Kernels &
Host Program
Generator
Device
Specific
Optimizer
Feedback
Tuning
Configuration
Generator
Search Space
Pruner
OpenARC
Auto-Tuner
Output Codes
Kernels for
Target Devices
Host Program
CUDA, OpenCL
Libraries
HeteroIR Common Runtime
with Tuning Engine
CUDA
GPU
AMD
GPU
Xeon
Phi
Altera
FPGA
Vetter, 27 Apr 2016

Informed Offloading Execution



Matrix Multiply



S. Lee, J.S. Meredith, and J.S. Vetter, “COMPASS: A Framework for Automated Performance Modeling and Prediction,” in *Proceedings of the 29th ACM on International Conference on Supercomputing. Newport Beach, California, USA: ACM, 2015, pp. 405-14.*

LULESH: Runtime and Working Set Size Predictions

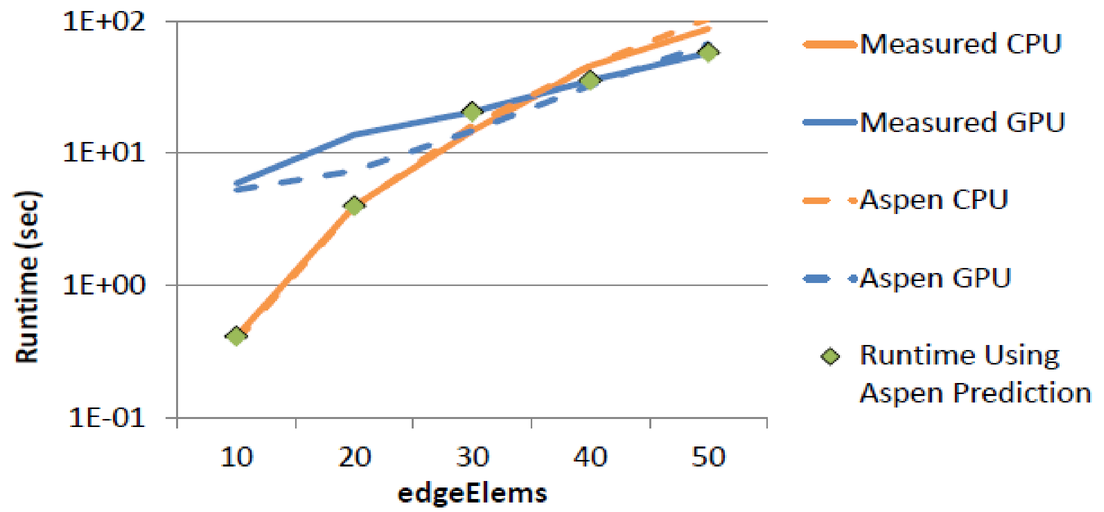


Fig. 7: Measured and predicted runtime of the entire LULESH program on CPU and GPU, including measured runtimes using the automatically predicted optimal target device at each size.

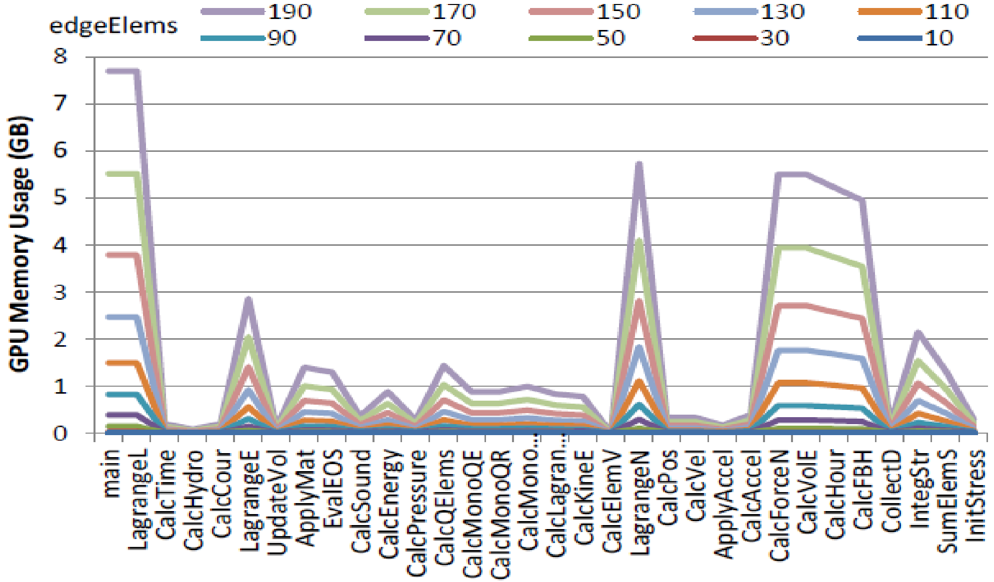
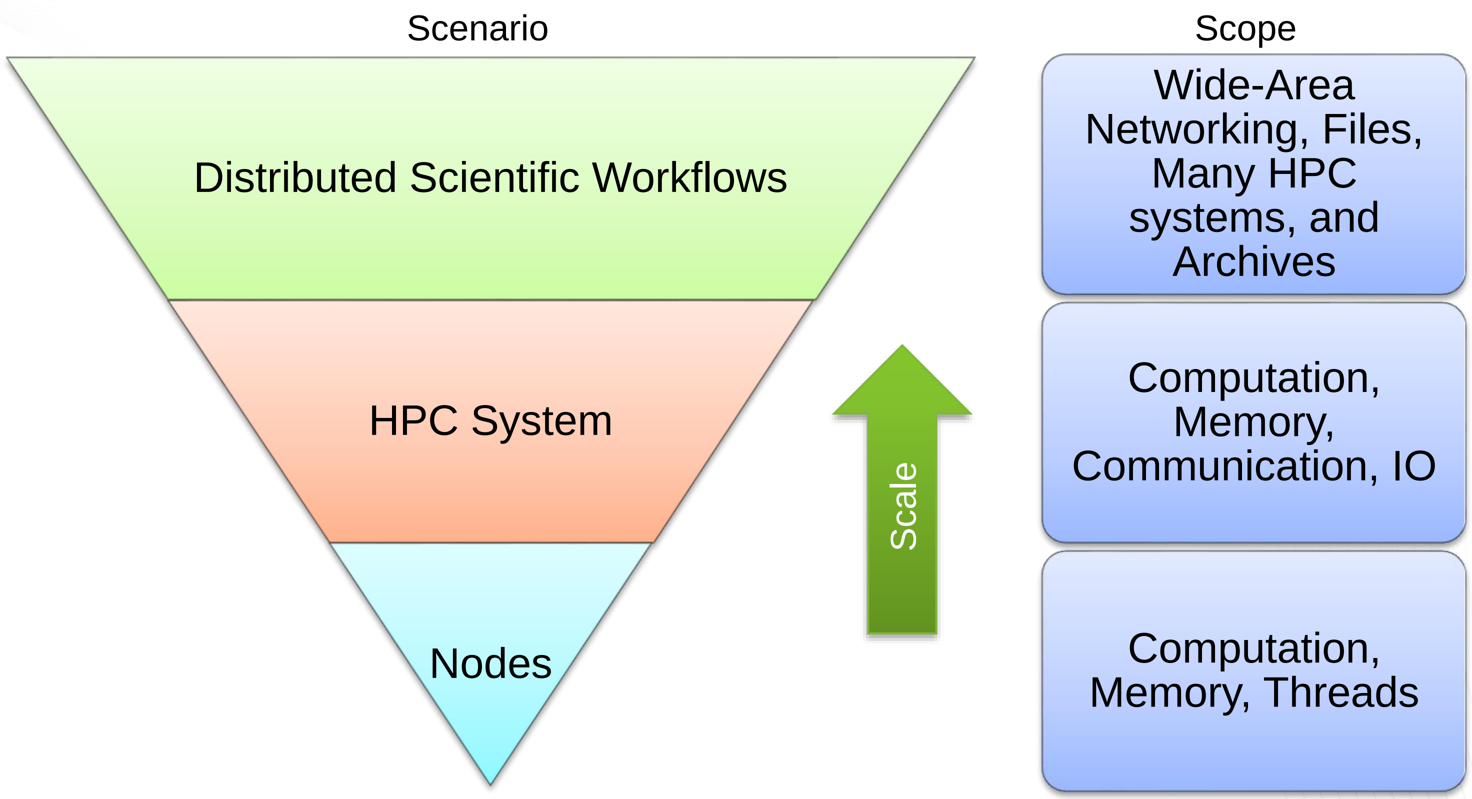


Fig. 8: GPU Memory Usage of each Function in LULESH, where the memory usage of a function is inclusive; value for a parent function includes data accessed by its child functions in the call graph.

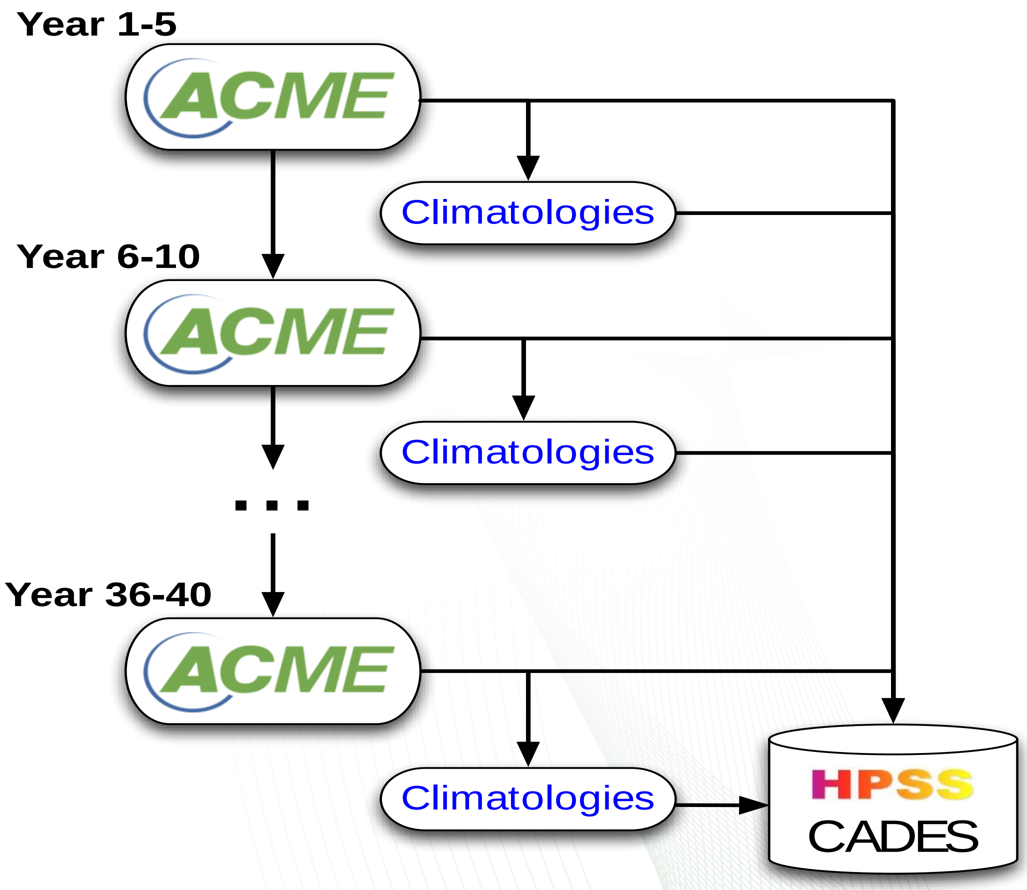
Using Aspen for Distributed Workflows

Aspen allows Multiresolution Modeling



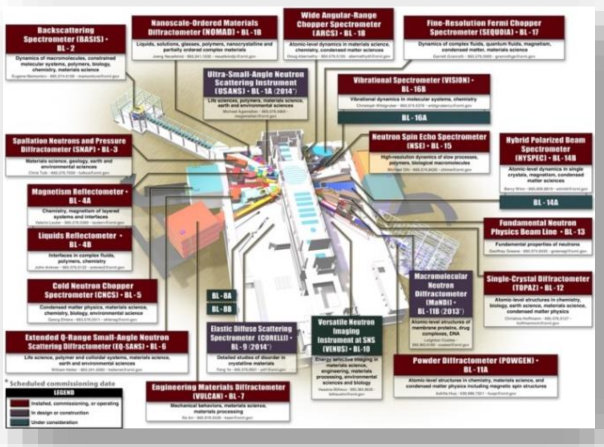
Simulation: ACME Workflows

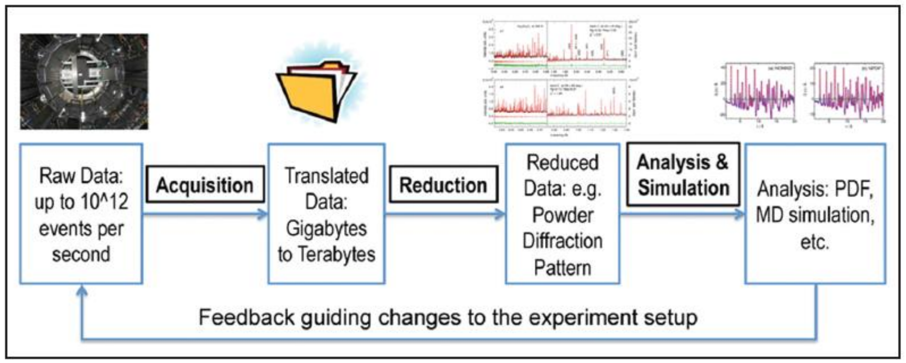
- Accelerated Climate Modeling for Energy (ACME)
- Coupled climate models with ocean, land, atmosphere and ice
- Climatologies and diagnostics give summaries of data
- Each stage of the workflow runs the ACME model for a few timesteps—helps keep simulations within batch queue limits
- Running on Hopper @ NERSC and Titan @ OLCF

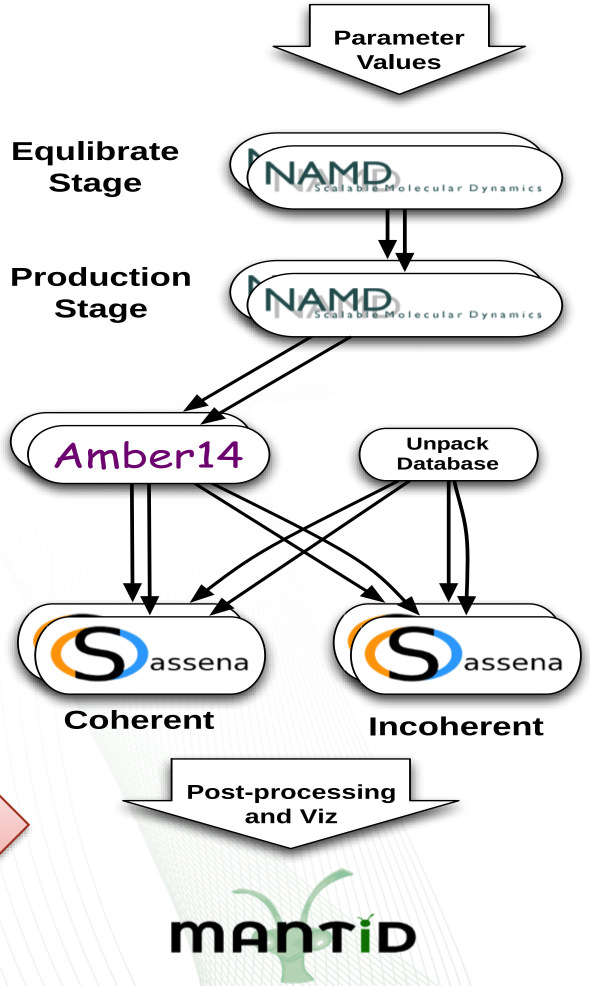


Experimental Data: SNS Workflows

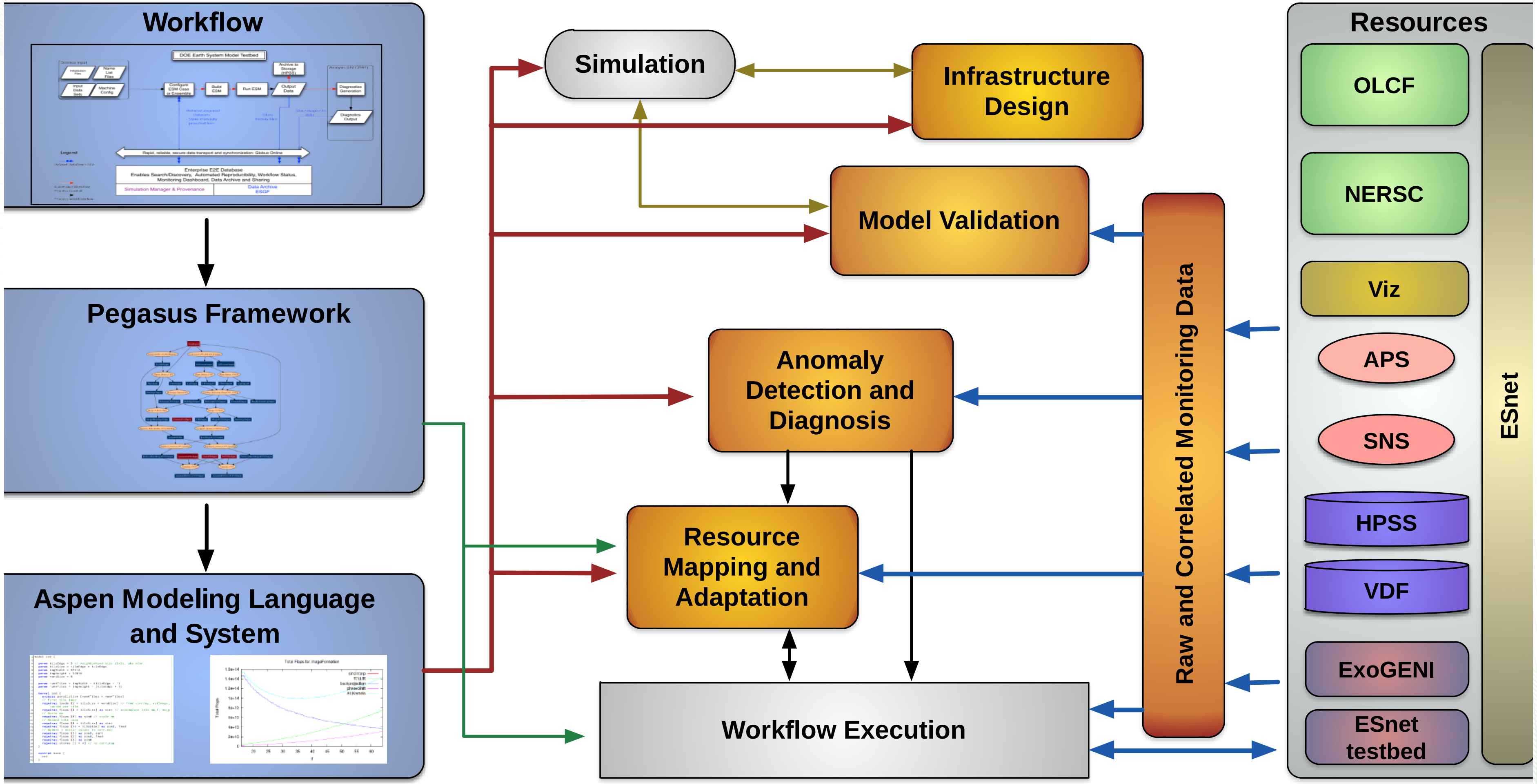
- Spallation Neutron Source
- Parameter sweep of molecular dynamics and neutron scattering
- Used to identify parameters that fit experimental data from SNS
- Currently being used for real science problems
- Large runs use 20 parameter values and require ~400,000 CPU hours
- Running on Hopper @ NERSC and coming to Titan @ ORNL







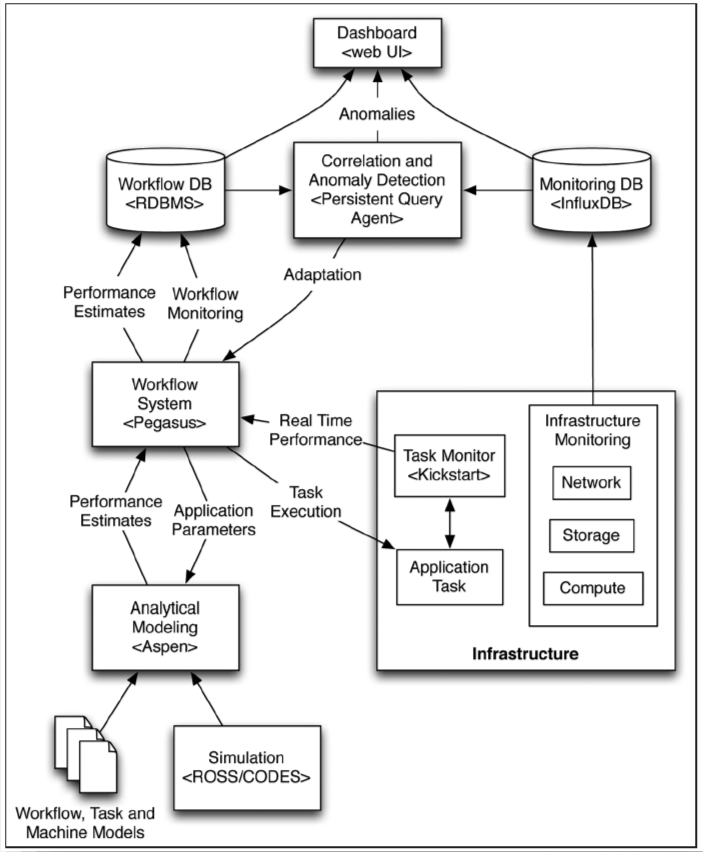
PANORAMA Overview



Automatically Generate Aspen from Pegasus DAX; Use Aspen Predictions to Inform/Monitor Decisions

```
kernel main
{
    par {
        seq {
            call namd_eq_200()
            call namd_prod_200()
        }
        seq {
            call namd_eq_290()
            call namd_prod_290()
        }
    }
    par {
        call unpack_database()
        call ptraj_200()
        call ptraj_290()
    }
    par {
        call sassena_incoh_200()
        call sassena_coh_200()
        call sassena_incoh_290()
        call sassena_coh_290()
    }
}
```

Listing 1: Automatically generated Aspen model for xample SNS workflow.



Executive Summary

- Both architectures and applications are growing more complex
  - Trends dictate that this will get worse, not better
  - This complexity creates irregularity in computation, communication, and data movement
- Dynamic resource management is one way to help manage this irregularity
  - Need accurate policies to guide resource decisions
  - Examples: greedy work stealing, algorithmic, historical, cost models, application specific, etc
- Posit that we can use application-specific performance models to inform scheduling decisions
  - Aspen performance modeling language helps create models
  - Two recent experiments
    - GPU offload
    - *Distributed scientific workflows*
- *Q&A*
  - *Can Aspen be accurate enough for these dynamic decisions?*
  - *How can we employ/influence the RT/OS in this process?*

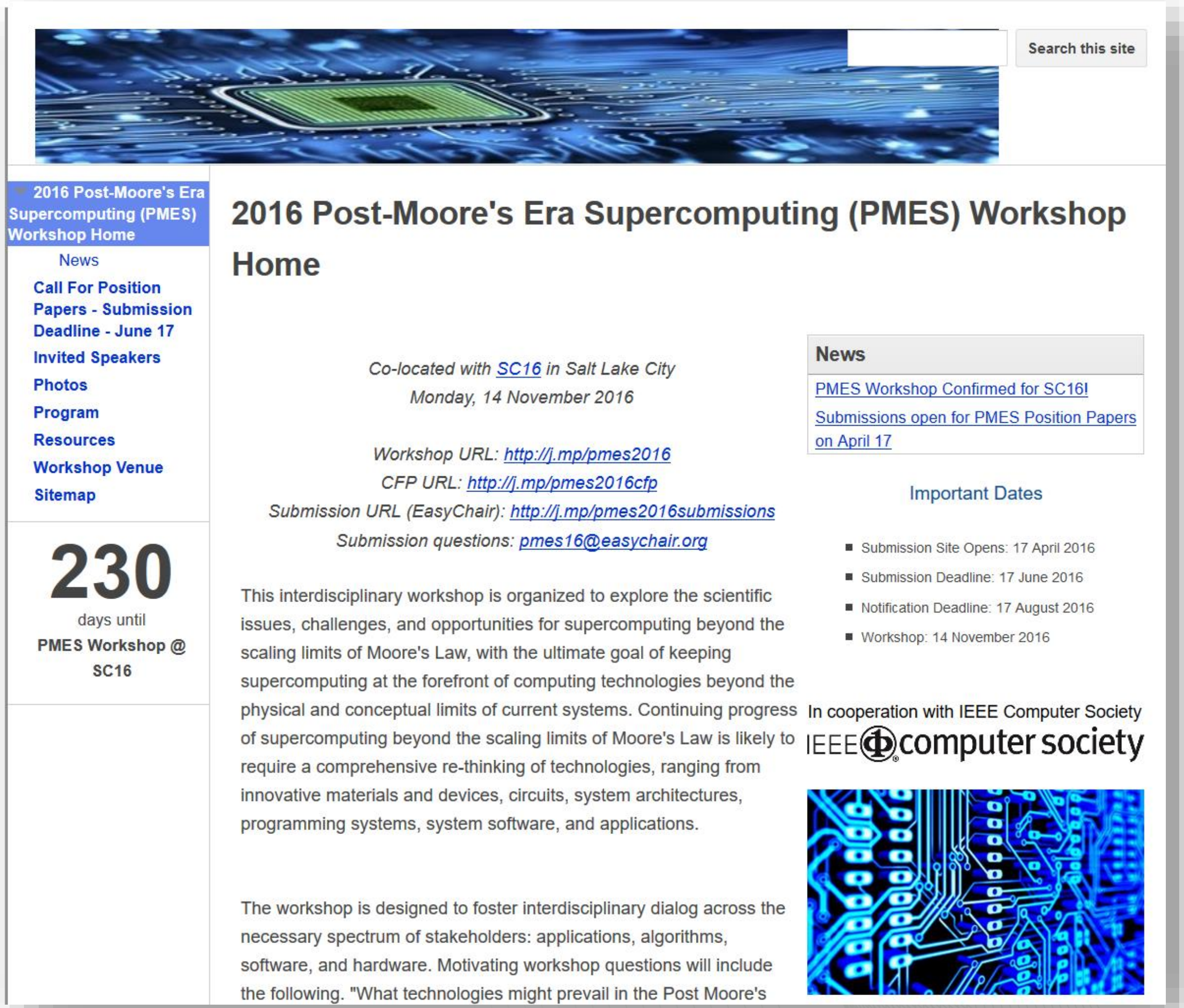
Acknowledgements

- Contributors and Sponsors
  - Future Technologies Group: http://ft.ornl.gov
  - US Department of Energy Office of Science
    - DOE Vancouver Project: https://ft.ornl.gov/trac/vancouver
    - DOE Blackcomb Project: https://ft.ornl.gov/trac/blackcomb
    - DOE ExMatEx Codesign Center: http://codesign.lanl.gov
    - DOE Cesar Codesign Center: http://cesar.mcs.anl.gov/
    - DOE Exascale Efforts: http://science.energy.gov/ascr/research/computer-science/
  - Scalable Heterogeneous Computing Benchmark team: http://bit.ly/shocmarx
  - US National Science Foundation Keeneland Project: http://keeneland.gatech.edu
  - US DARPA
  - NVIDIA CUDA Center of Excellence

OAK RIDGE National Laboratory

PMES Workshop @ SC16

- https://j.mp/pmes2016
- @SC16
- Position papers due June 17



Oak Ridge National Laboratory