Autotuning and ECE Computing Infrastructure

18-613: Foundations of Computer Systems
9th Lecture, **April 18, 2019**

Instructor:

Franz Franchetti

Outline

- **Autotuning**
- ECE software
- Andrew software
- PSC software
- Computing infrastructure

Programmer Demographics

Black Belt Programmer

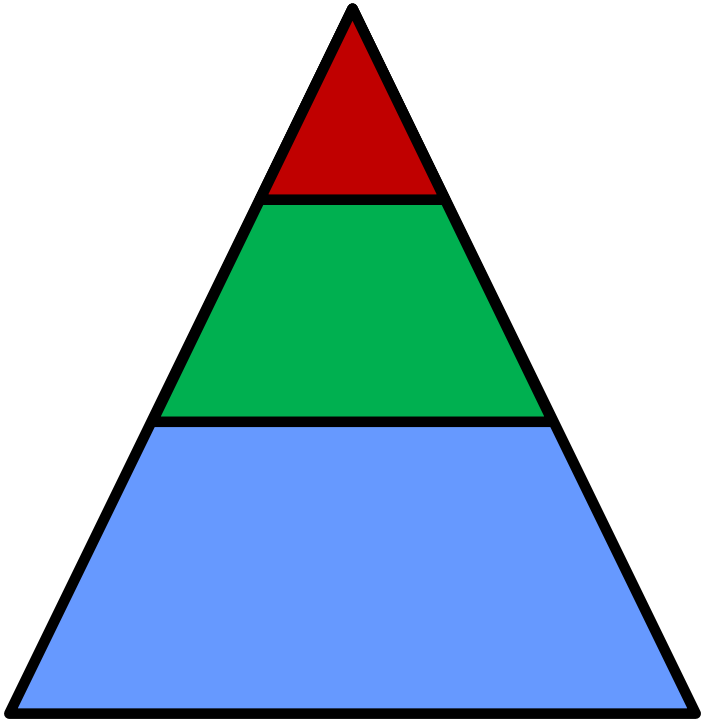
- Performance libraries, device drivers
- C, Assembly, low-level tricks
- ***Goal: performance above all***

Software engineers

- Large applications
- C++, Java, C#
- ***Goal: Maintenance vs. performance***

Average programmers

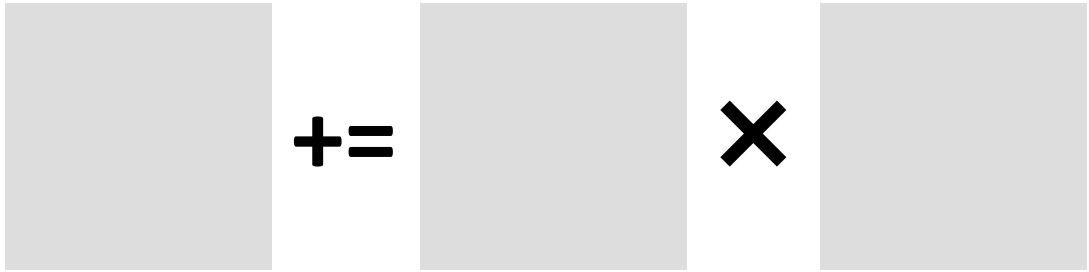
- Small ad-hoc applications
- C, C++, Java, Perl, Python
- ***Goal: implement their algorithm quickly***



Compilers: The Fundamental Problem

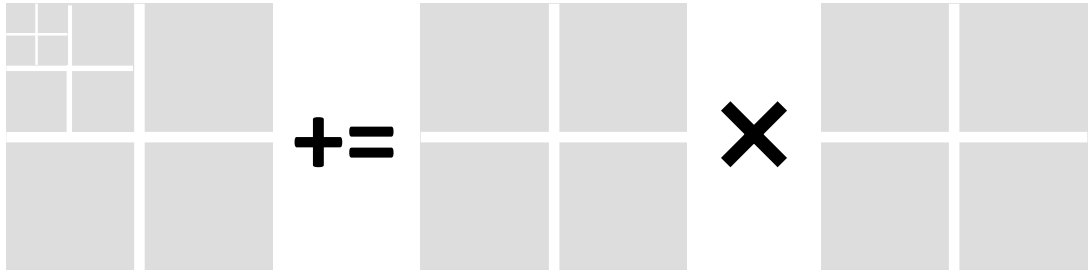
Matrix-matrix multiplication

```
for i=1:N
  for j=1:N
    for k=1:N
      C[i,j] += A[i,k]*B[k,j]
```



Tiled matrix-matrix multiplication

```
for i=1:NB:N
  for j=1:NB:N
    for k=1:NB:N
      for i0=i:NU:i+NB
        for j0=j:NU:j+NB
          for k0=k:NU:k+NB
            for k00=k0:1:k0+NU
              for j00=j0:1:j0+NU
                for i00=i0:1:i0+NU
                  C[i00,j00] +=
                    A[i00,k00]*B[k00,j00]
```

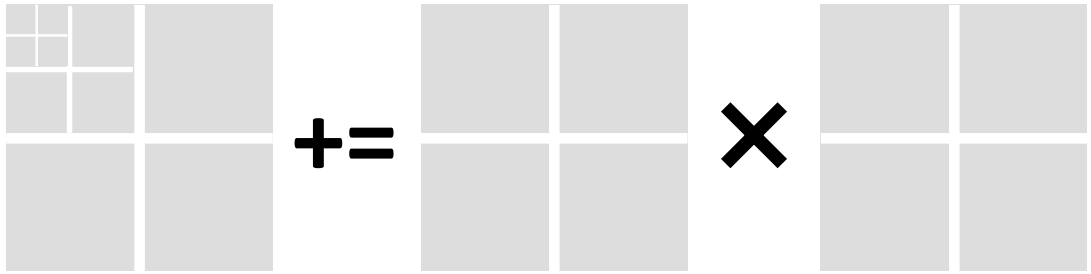


Problem: which transformation order? What parameter values?

How to Overcome Compiler Limitations?

- **Autotuning**

  Write parameterized program and empirically find good parameters



Parameter: tile size, levels

- **Program generation**

  Automatically generate big basic blocks and special case code

```
for (i = 0; i < 4; i++)
   A[i] = B[i] * C[i];
```

```
A[0] = B[0] * C[0];
A[1] = B[1] * C[1];
A[2] = B[2] * C[2];
A[3] = B[3] * C[3];
```

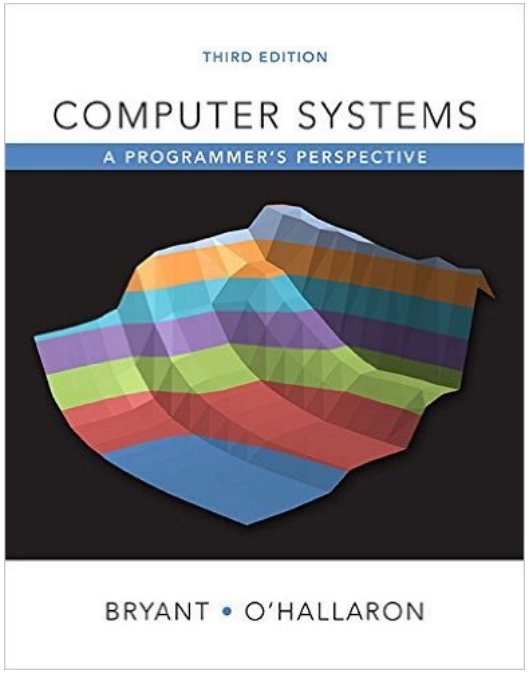
Parameter: which loop, unroll factor

Autotuning: From Simple to "Really Hard"

- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming
- **Level 2: scripting for code specialization**
  text-based program generation, e.g., ATLAS
- **Level 3: add compiler technology**
  internal code representation, e.g., FFTW's genfft
- **Level 4: synthesize the program from scratch**
  high level representation, e.g., TCE and Spiral

Step 1: Basic C Level Optimizations

- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming



http://www.cs.cmu.edu/afs/cs/academic/class/15213-s12/www/

Unrolling & Accumulating: Double *

- **Case**
  - Intel Haswell
  - Double FP Multiplication
  - Latency bound: 5.00. Throughput bound: 0.50

FP *	Unrolling Factor L							
K	1	2	3	4	6	8	10	12
1	5.01	5.01	5.01	5.01	5.01	5.01	5.01	
2		2.51		2.51		2.51		
3			1.67					
4				1.25		1.26		
6					0.84			0.88
8						0.63		
10							0.51	
12								0.52

Accumulators

Autotuning: From Simple to "Really Hard"

- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming
- **Level 2: scripting for code specialization**
  text-based program generation, e.g., ATLAS
- **Level 3: add compiler technology**
  internal code representation, e.g., FFTW's genfft
- **Level 4: synthesize the program from scratch**
  high level representation, e.g., TCE and Spiral

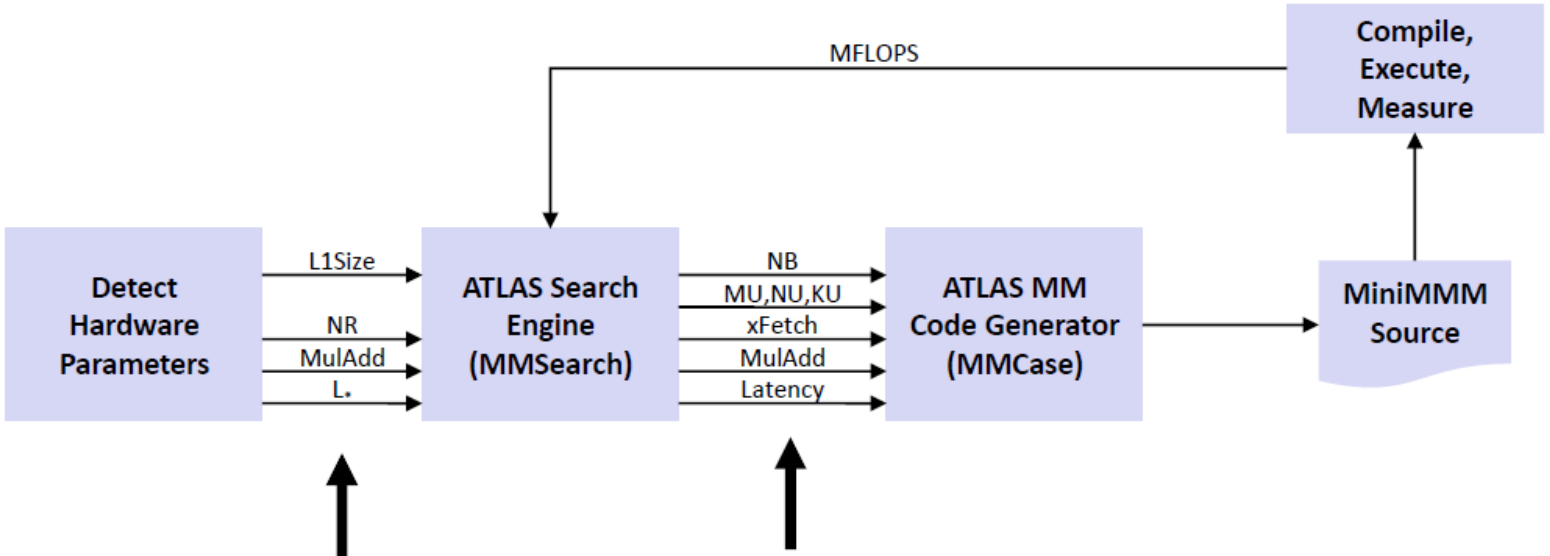
ATLAS: Code Generation for MMM

- Successor of PhiPAC, BLAS program generator (web)
- Idea: automatic porting

- People can also contribute handwritten code
- The generator uses empirical search over implementation alternatives to find the fastest implementation
  no vectorization or parallelization: so not really used anymore
- We focus on BLAS3 MMM
- Search only over cost $2n^3$ algorithms
  (cost equal to triple loop)

Source: Markus Püschel, ETH Zürich: How to Write Fast Numerical Code, 2011
https://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring11/course.html

ATLAS: Architecture



Search parameters:

- span search space
- specify code
- found by orthogonal line search

Hardware parameters:

- L1Size: size of L1 data cache
- NR: number of registers
- MulAdd: fused multiply-add available?
- L_* : latency of FP multiplication

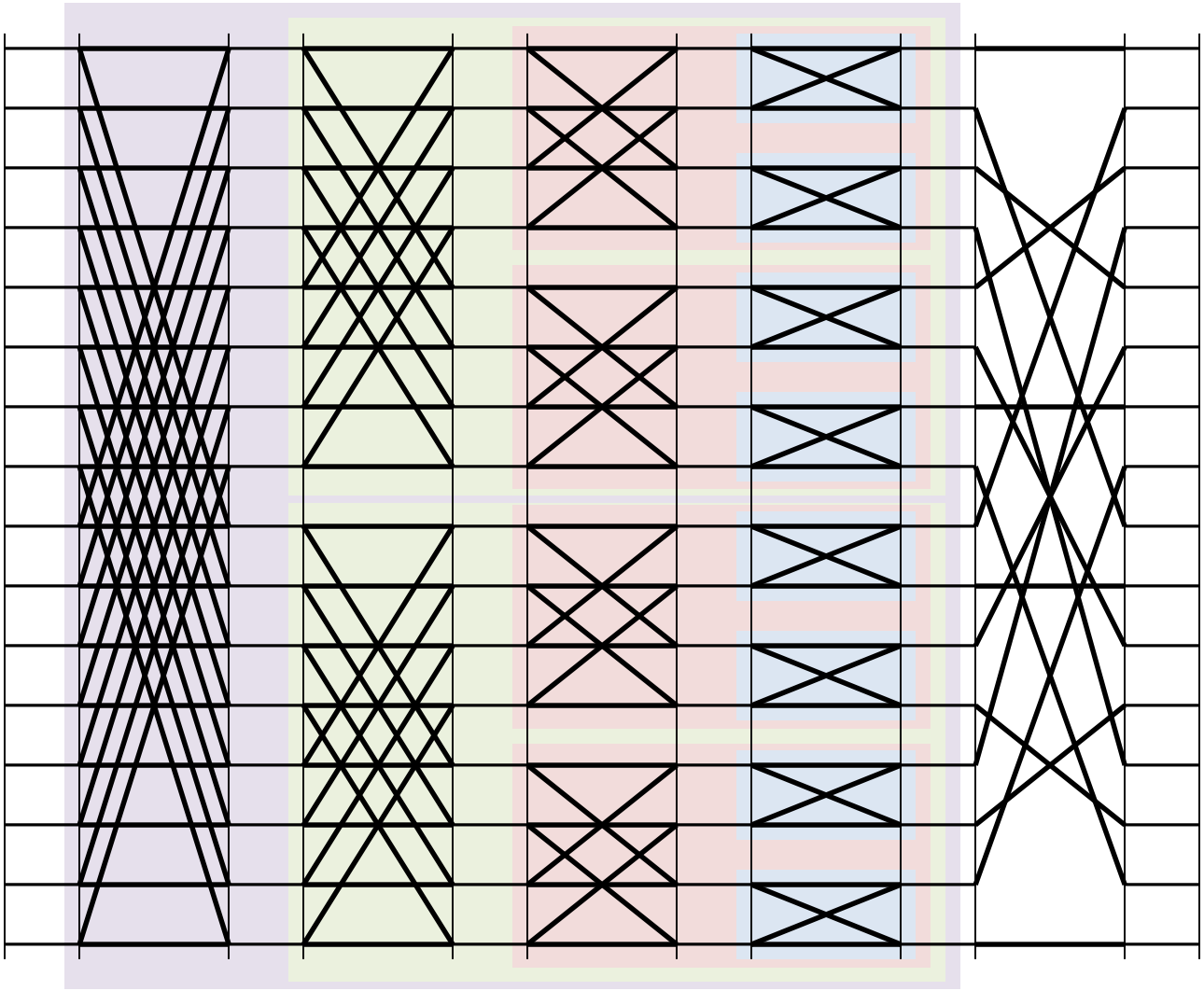
source: Pingali, Yotov, Cornell U.

Source: Markus Püschel, ETH Zürich: How to Write Fast Numerical Code, 2011
https://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring11/course.html

Autotuning: From Simple to "Really Hard"

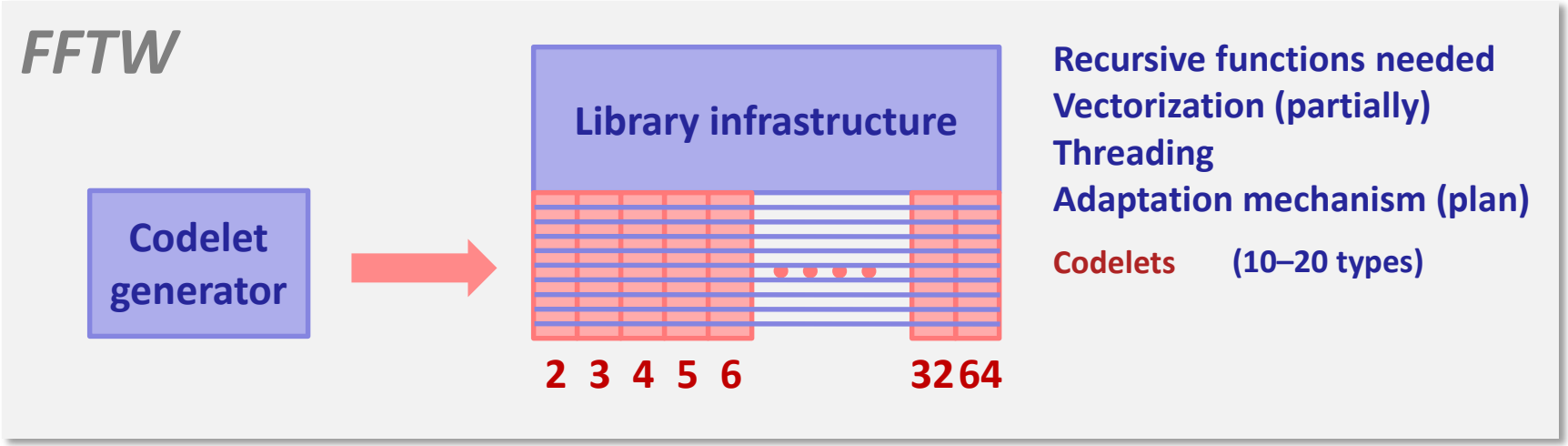
- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming
- **Level 2: scripting for code specialization**
  text-based program generation, e.g., ATLAS
- **Level 3: add compiler technology**
  internal code representation, e.g., FFTW's genfft
- **Level 4: synthesize the program from scratch**
  high level representation, e.g., TCE and Spiral

Example FFT: Recursive FFT Algorithm



FFTW: Compiler and Autotuner for FFTs

- Choice of algorithm
- Locality optimization
- Constants
- Fast basic blocks
- Adaptivity



Source: Markus Püschel, ETH Zürich: How to Write Fast Numerical Code, 2011
https://www.inf.ethz.ch/personal/markusp/teaching/263-2300-ETH-spring11/course.html

FFTW: genfft and the Cooley Tukey FFT

DSP book:

$$y_k = \sum_{j=0}^{n-1} x_j \omega_n^{jk} = \sum_{j_2=0}^{p-1} \left[\left(\sum_{j_1=0}^{q-1} x_{pj_1+j_2} \omega_q^{j_1 k_1} \right) \omega_n^{j_2 k_1} \right] \omega_p^{j_2 k_2},$$

where $n = pq$ and $k = k_1 + qk_2$.

OCaml code:

```
let cooley_tukey n p q x =
  let inner j2 = fftgen q
    (fun j1 -> x (p * j1 + j2)) in
  let twiddle k1 j2 =
    (omega n (j2 * k1)) @* (inner j2 k1) in
  let outer k1 = fftgen p (twiddle k1) in
   (fun k -> outer (k mod q) (k / q))
```

Source: Matteo Frigo: "A Fast Fourier Transform Compiler," In Proceedings ACM PLDI 1999
http://www.fftw.org/pldi99-slides.ps.gz

FFTW: genfft at Work

2-point FFT

```
tmp1 = REAL(input[0]);
tmp5 = REAL(input[0]);
tmp6 = IMAG(input[0]);
tmp2 = IMAG(input[0]);
tmp3 = REAL(input[1]);
tmp7 = REAL(input[1]);
tmp8 = IMAG(input[1]);
tmp4 = IMAG(input[1]);
REAL(output[0]) = ((1 * tmp1) - (0 * tmp2))
      + ((1 * tmp3) - (0 * tmp4));
IMAG(output[0]) = ((1 * tmp2) + (0 * tmp1))
      + ((1 * tmp4) + (0 * tmp3));
REAL(output[1]) = ((1 * tmp5) - (0 * tmp6))
      + ((-1 * tmp7) - (0 * tmp8));
IMAG(output[1]) = ((1 * tmp6) + (0 * tmp5))
      + ((-1 * tmp8) + (0 * tmp7));
```

Algebraic Simplification

```
let rec algsimpM x =
  memoizing
    (function
        Num a -> snumM a
      | Plus a ->
          mapM algsimpM a >>= splusM
      | Times (a, b) ->
          algsimpM a >>= fun a' ->
            algsimpM b >>= fun b' ->
              stimesM (a', b')
      | Uminus a ->
          algsimpM a >>= suminusM
      | Store (v, a) ->
          algsimpM a >>= fun a' ->
            returnM (Store (v, a'))
      | x -> returnM x)
    x
```

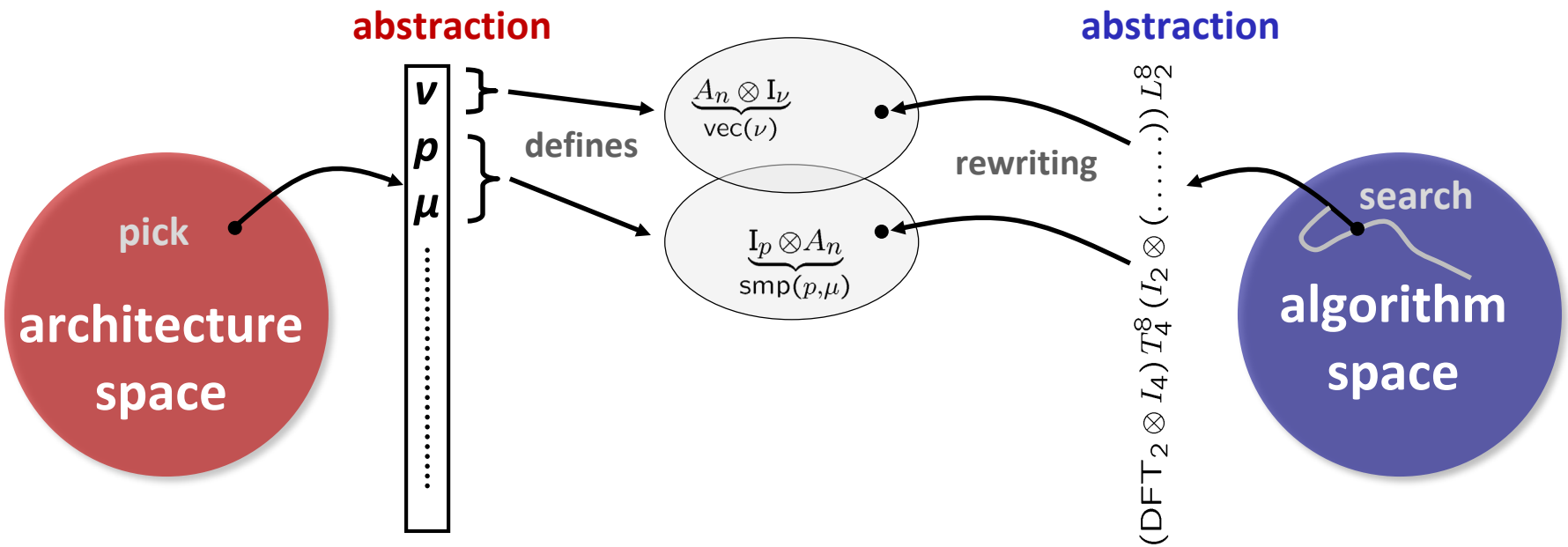
Source: Matteo Frigo: "A Fast Fourier Transform Compiler," In Proceedings ACM PLDI 1999
http://www.fftw.org/pldi99.pdf

Autotuning: From Simple to "Really Hard"

- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming
- **Level 2: scripting for code specialization**
  text-based program generation, e.g., ATLAS
- **Level 3: add compiler technology**
  internal code representation, e.g., FFTW's genfft
- **Level 4: synthesize the program from scratch**
  high level representation, e.g., TCE and Spiral

Spiral's Domain-Specific Program Synthesis

Model: common abstraction
= spaces of matching formulas



Architectural parameter:
Vector length,
#processors, …

optimization

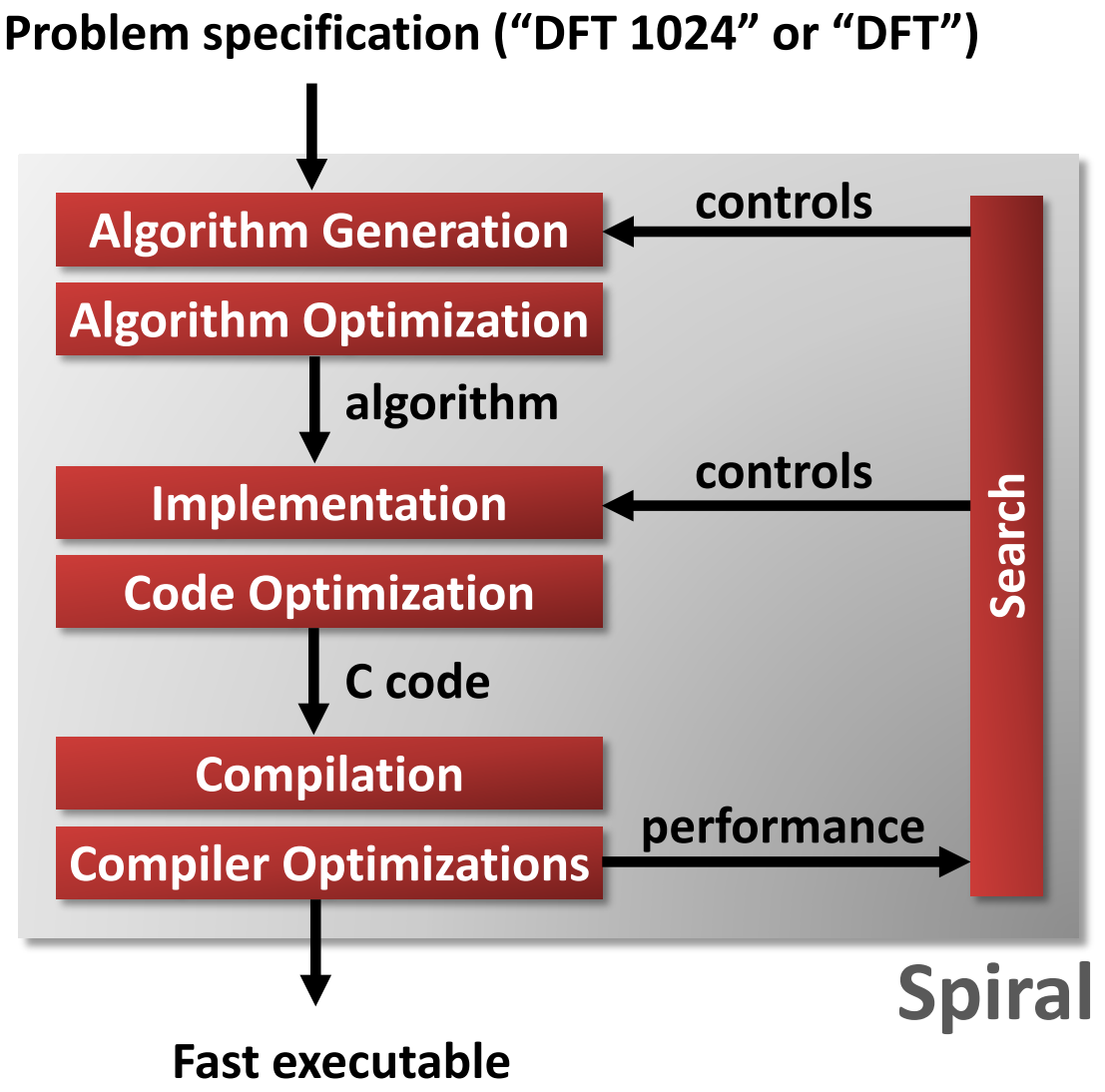
Kernel:
problem size,
algorithm choice

How Spiral Works

Complete automation of the implementation and optimization task

Basic ideas:

- ***Declarative representation*** of algorithms
- ***Rewriting systems*** to generate and optimize algorithms at a high level of abstraction



Autotuning: From Simple to "Really Hard"

- **Level 0: simple C program**
  implements the algorithm cleanly
- **Level 1: C macros plus search script**
  use C preprocessor for meta-programming
- **Level 2: scripting for code specialization**
  text-based program generation, e.g., ATLAS
- **Level 3: add compiler technology**
  internal code representation, e.g., FFTW's genfft
- **Level 4: synthesize the program from scratch**
  high level representation, e.g., TCE and Spiral

Outline

- Autotuning
- **ECE software**
- Andrew software
- PSC software
- Computing infrastructure

ANSYS

Careers Academic Customer Portal Resource Library Follow 5.6k

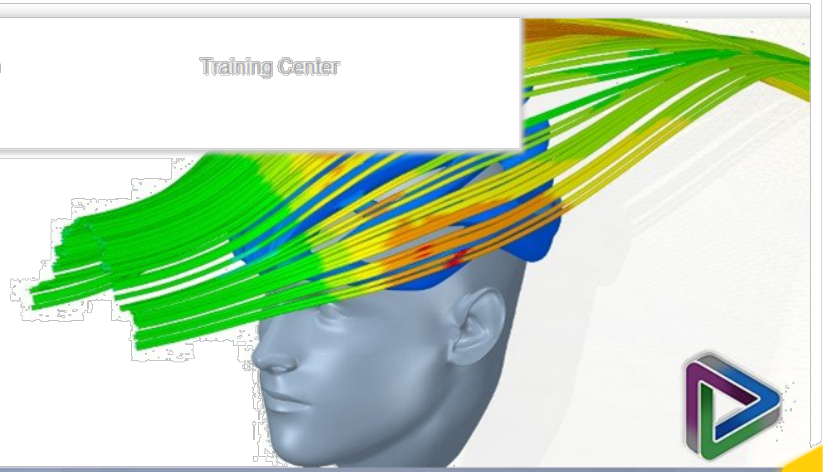
Products Solutions Services Support About ANSYS search

Consulting Learning Hub Training Center

ANSYS Discovery Live

The new technology that will
change product design forever

Download Now >



CONTACT

Explore Pervasive Engineering Simulation

Discover how engineering simulation is expanding across the entire product lifecycle, from digital exploration to digital prototyping to operations and maintenance using digital twins.

Select a physics area to learn more about what sets ANSYS software apart from other engineering simulation tools.

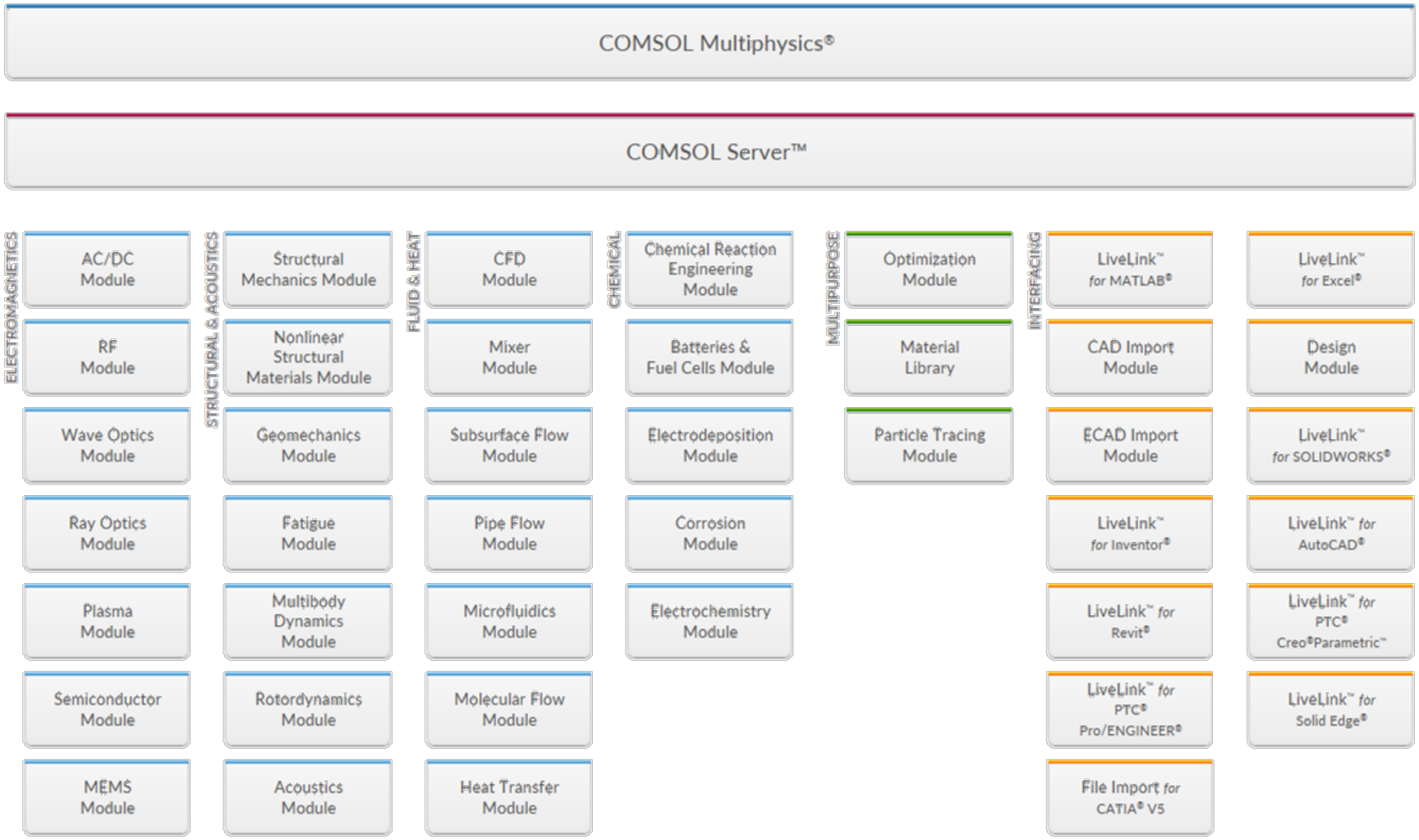
Structures
Fluids
Electromagnetics
Semiconductors
ANSYS
Embedded Software
Systems
Partner Network
3-D Design
Platform
Multiphysics Meshing Optimization Customization
Data and Process Management Cloud HPC

http://www.ansys.com/

Cadence

https://www.cadence.com/

Multiphysics Software Product Suite

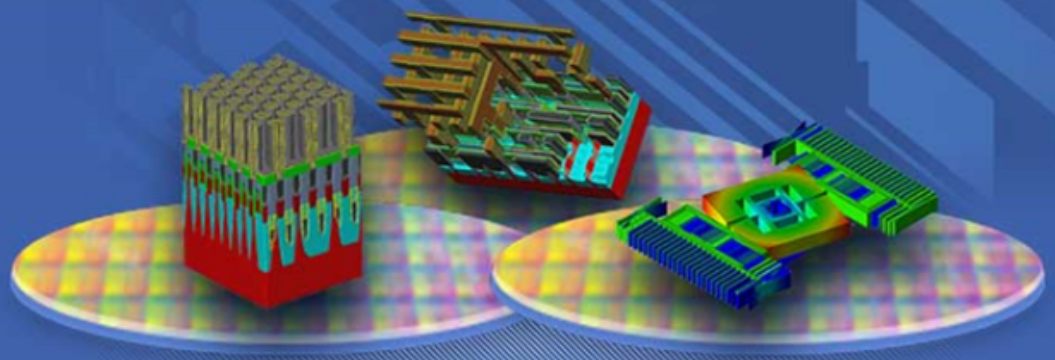


https://www.comsol.com/

COVENTOR
A Lam Research Company

search

ABOUT US | SEMICONDUCTOR SOLUTIONS | MEMS SOLUTIONS | SUPPORT | NEWS & EVENTS | CONTACT US



Predicting Actual from Virtual

Solving process modeling, design automation, and integration challenges for the semiconductor and MEMS industries.

What's New

Video Interview: Problems and Solutions at 7nm

LEARN MORE

White Paper: Semiconductor Process Variation

LEARN MORE

Semiconductor Solutions

Develop and optimize manufacturing process flows, reducing time consuming and costly experimental learning cycles.

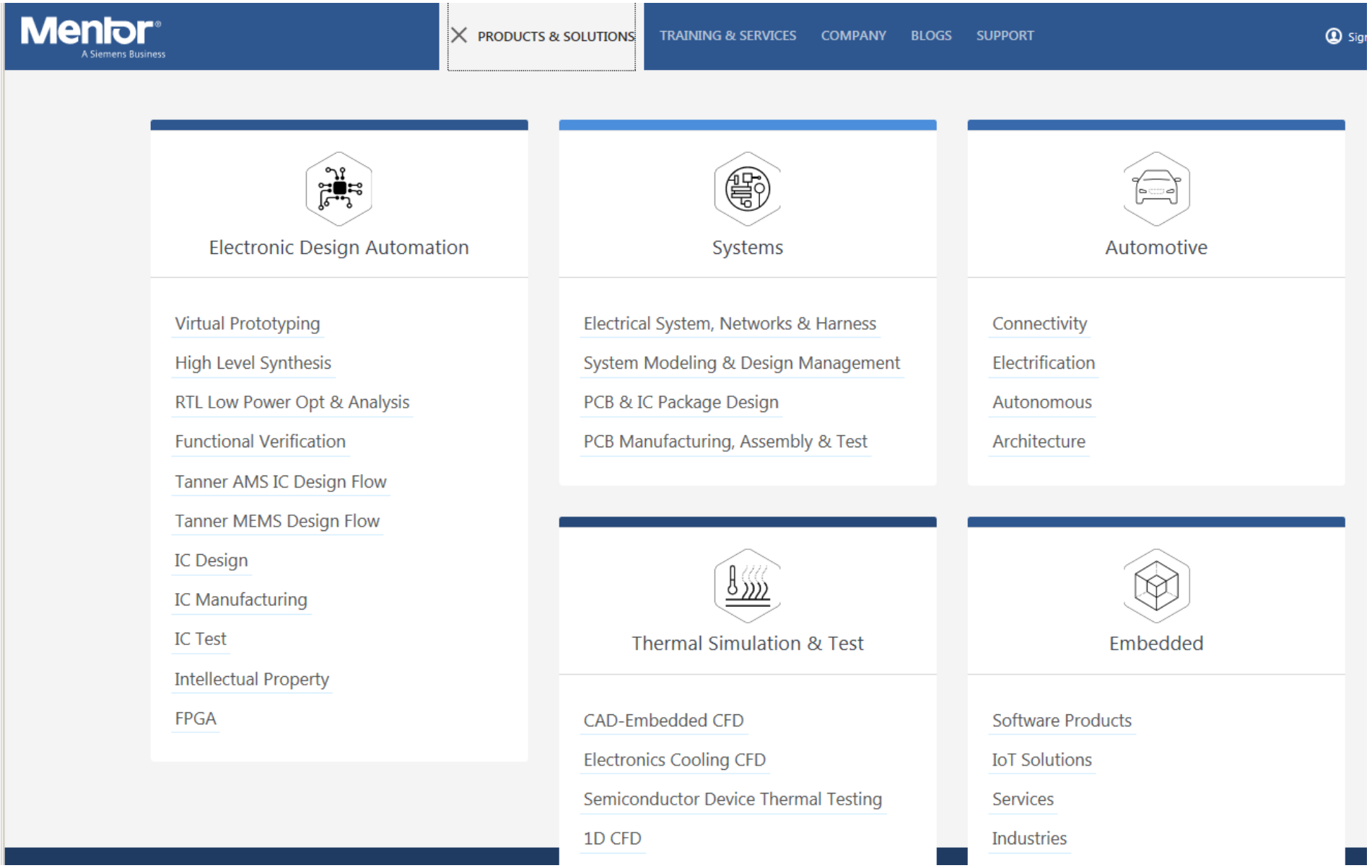
LEARN MORE »

MEMS Solutions

Design and integrate MEMS and IoT devices significantly faster, without developing and maintaining custom models.

LEARN MORE »

http://www.coventor.com/



https://www.mentor.com/

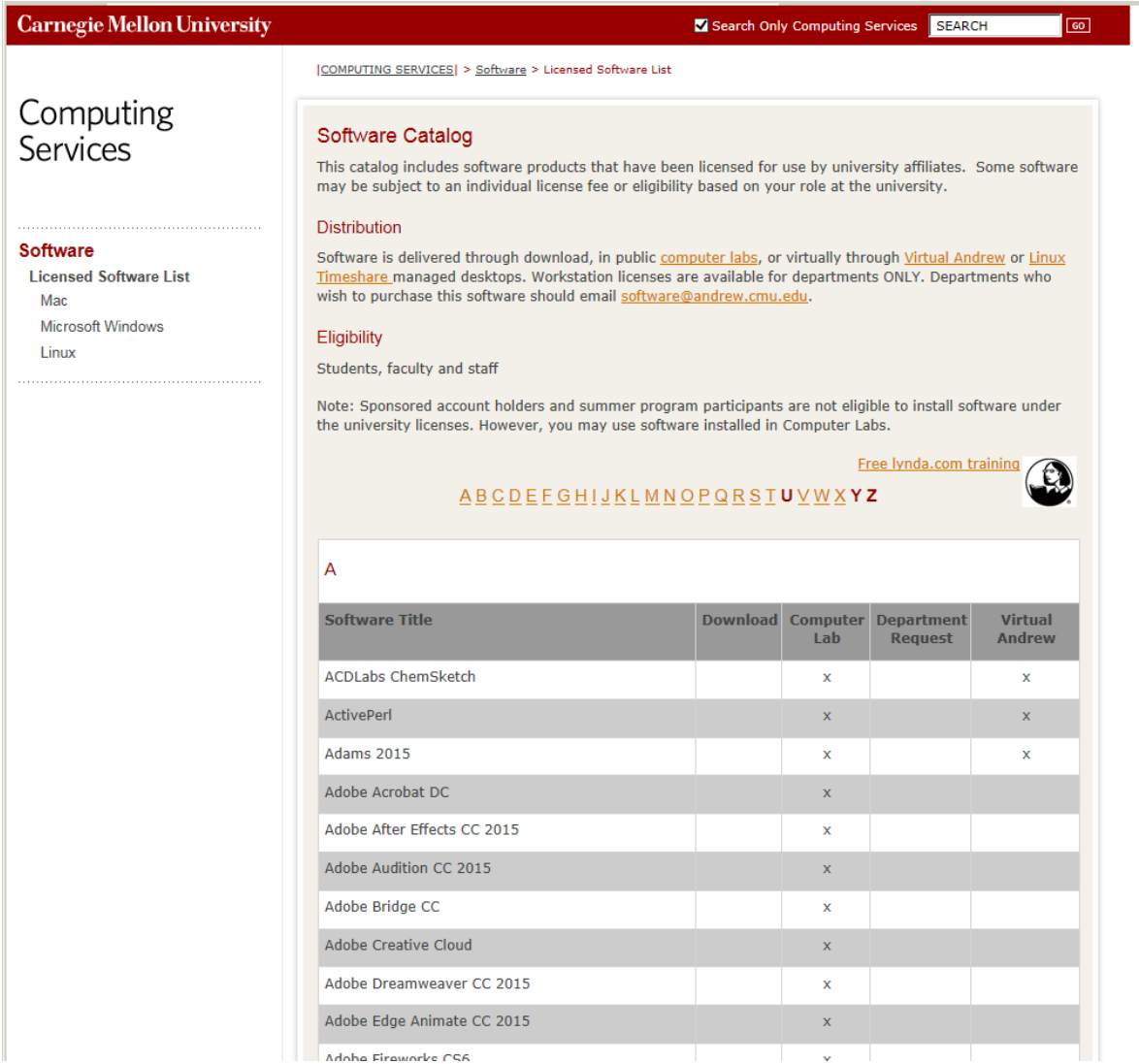
Outline

- Autotuning
- ECE software
- **Andrew software**
- PSC software
- Computing infrastructure

Andrew Software

https://www.cmu.edu/computing/software/all/

- SPSS
- IMSL
- Maple
- Mathematica
- Matlab
- SAS
- S-PLUS
- Wolfram Alpha
- R/R Studio
- Python
- MySQL, MS SQL Server, PostgreSQL



Outline

- Autotuning
- ECE software
- Andrew software
- **PSC software**
- Computing infrastructure

PSC Software

- Abaqus (finite elements)
- Abinit (density functional theory)
- Anaconda (data science platform)
- ANSYS
- BLAST (protein sequence alignment)
- Caffe (deep learning)
- CFOUR (computational chemistry)
- Chainer (neural nets)
- Desmond (molecular dynamics)
- DReAMM (MCell simulation)

PSC

https://www.psc.edu/resources/software

Software installed on PSC systems

The Module package

The environment management package Module is essential for running software on PSC systems. Be sure to check if there is a module for the software you want to use by typing `module avail software-name`

The `module help` command lists any additional modules that must also be loaded. Note that in some cases the order in which these additional modules are loaded matters.

Documentation on the module command.

You may request that additional software be installed by sending mail to remarks@psc.edu.

See software by category

See software by resource

Clear selection

Software	Description	Computing Resource
Abaqus	Suite for finite element analysis and computer-aided engineering	Bridges
Abinit	Calculates electronic density and	Bridges

PSC Software (2)

- GAMESS (ab initio quantum chemistry)
- Gurobi (optimization system)
- Julia (programming language)
- Keras (neural networks, on top of Theano and TensorFlow)
- LevelDB (key-value store)
- LS-DYNA (finite element program)
- Mathematica, Matlab, R
- MCell (Monte Carlo cell simulation)
- MCT (library for creating coupled models)
- MUMPS (solver for large linear systems)
- SuperLU (solves sparse linear systems)

PSC Software (3)

- NAMD, NWCHEM, LAMMPS, CHARMM, CFOUR, Quantum Espresso, SIESTA,
- OpenCV (computer vision)
- PETSc (PDE solver)
- Python, Spark, TensorFlow, Theano, Torch
- PyTorch (deep learning)
- Scikit-learn (data mining)
- SCOTCH (graph partitioning and clustering)

Outline

- Autotuning
- ECE software
- Andrew software
- PSC software
- **Computing infrastructure**

Outline

- Personal Computing
- CMU Andrew Computing
- ECE Physical Spaces
- ECE Compute Clusters
- ECE Data Science Cloud
- ECE HTCondor
- CIT Partnerships
- PSC Bridges
- XSEDE Project
- Cloud Providers

Introductions

Dan Fassinger
Executive Manager
ECE IT Services

Jim McKinney
Data Center Manager
ECE IT Services

information technology services

Hours: M-F 8a-5p
Location:
Hamerschlag Hall A200 suites
Docs and resources:
https://userguide.its.cit.cmu.edu/
Email: help@ece.cmu.edu

Computing Options: **Personal Computing**

- **Personal computer (2018)**
  - Laptop or desktop
  - Intel Core i5/i7 w/ HT, 2-4 cores
  - 8GB to 64GB DDR4
  - 500GB SSD
  - Integrated graphics (Intel HD or Iris)
  - 1GB LAN / 802.11ac WLAN

Typical programs and problem sets

- Personal productivity, internet browser
- Development workflow tools
- Concept testing and simulation

Computing Options **CMU Andrew Computing**

- **Public Clusters**
  - Workstation or Desktop
  - Most University software installed
- **Virtual Andrew**
  - VMWare Horizon Client (VDI)
  - Cluster software accessible remotely
- **Linux Timeshares**
  - SSH and X-session
  - Restarted nightly

- **Ready to use**
- **Prepared and managed for you**
- **Good for productivity or moving small data**
- **Comparable or worse performance compared to personal laptop**

- **Campus Cloud**
  - SaaS, PaaS w/ fee ($$)
  - Guest VM instance with managed environment

- **Rental capacity**
- **Personal server w/o physical responsibility**

Computing Options: **ECE Physical Spaces**

- **Ugrad/grad labs**
  - HH 1303, 1204, A101, A104 – Equipment stations
- **Linux cluster/lab**
  - HH 1305
  - Workstation class systems
- **Capstone Lab**
  - HH 1307

- Teaching space (ex. 18-240 18-100)
- GPU SDK, LabView
- Console access for graphical Linux simulations (EDA and FEA tools)
- All require cardkey access

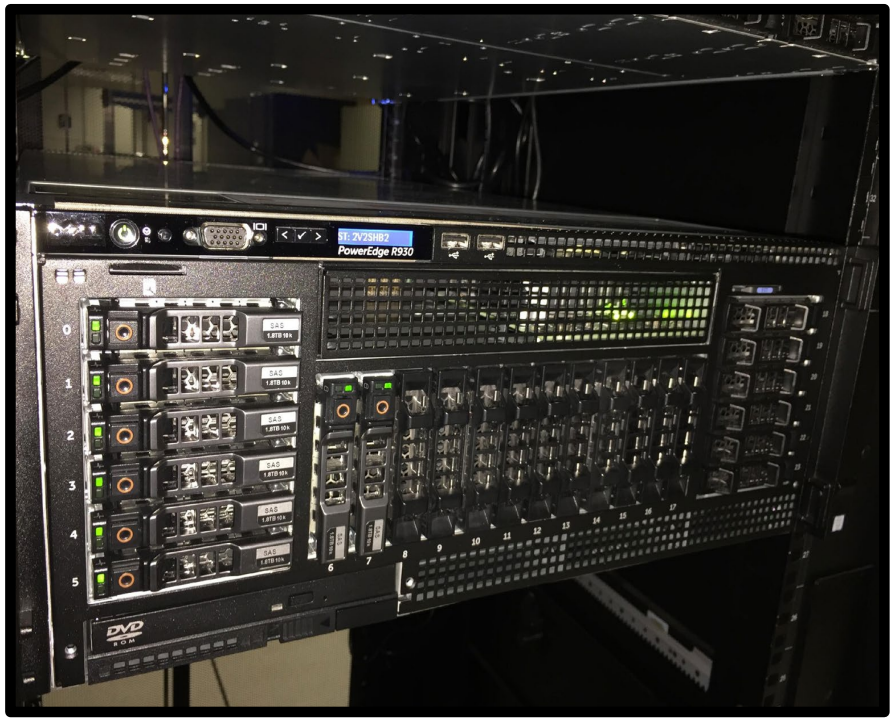
Computing Options: **ECE Compute Clusters**

- **Numbers cluster**
  - Ece[000:031].ece.local.cmu.edu
  - RedHat Enteprise Linux
  - Condor access
  - Engineering software
  - Shared storage
- **GUI cluster**
  - Ece-gui-[000-007].ece.cmu.edu
  - FastX or X-session for remote graphical access

- PowerEdge R430
- 2 x Intel Xeon E5-2640 v4, 2.4GHz, 8GT/s QPI, 10 cores w/ HT
- 128GB 2400MT/s RDIMM
- iSCSI link to SAN
- Housed in Cyert Data Center

Computing Options: **ECE Data Science Cloud**

- **Large memory and GPU**
  - Moderate sized simulations and parallel compute jobs
  - Customized environment
  - Fast storage access
  - 10GB uplinks



- PowerEdge R930
  - 4 x Intel Xeon E7-8870 v3, 9.6GT/s QPI, 18 cores w/ HT
  - 3TB 2133 MT/s RDIMM
- PowerEdge R730
  - 2 x Intel Xeon E5-2698 v3, 9.6GT/s QPI, 16 cores w/ HT
  - 768GB 2133MT/s RDIMM
  - 2 x Nvidia Tesla K80 GPU (x2)

Computing Options: **ECE HTCondor**

- Batch submission system
- Job queuing, scheduling
- Serial or parallel jobs
- Harness compute power of entire clusters

- **Access via ECE Numbers Cluster**
- **Submitting first Condor job**
  - Prepare
  - Compile
  - Submit
  - Retrieve
- **Use with Matlab, Comsol, Cadence, std. engineering simulations**

Computing Options: **CIT Partnerships**

- **Venkat Viswanathan**
  - Co-location in PSC Monroeville data center
  - 12 x HPE Blades
    - 12 x XL1x0r Gen9 Intel Xeon E5-2683v4, 16 core
      - 128GB DDR4 RAM
    - 48 x Nvidia Tesla K80
  - Managed by Venkat partnership group

Computing Options: **PSC Bridges**

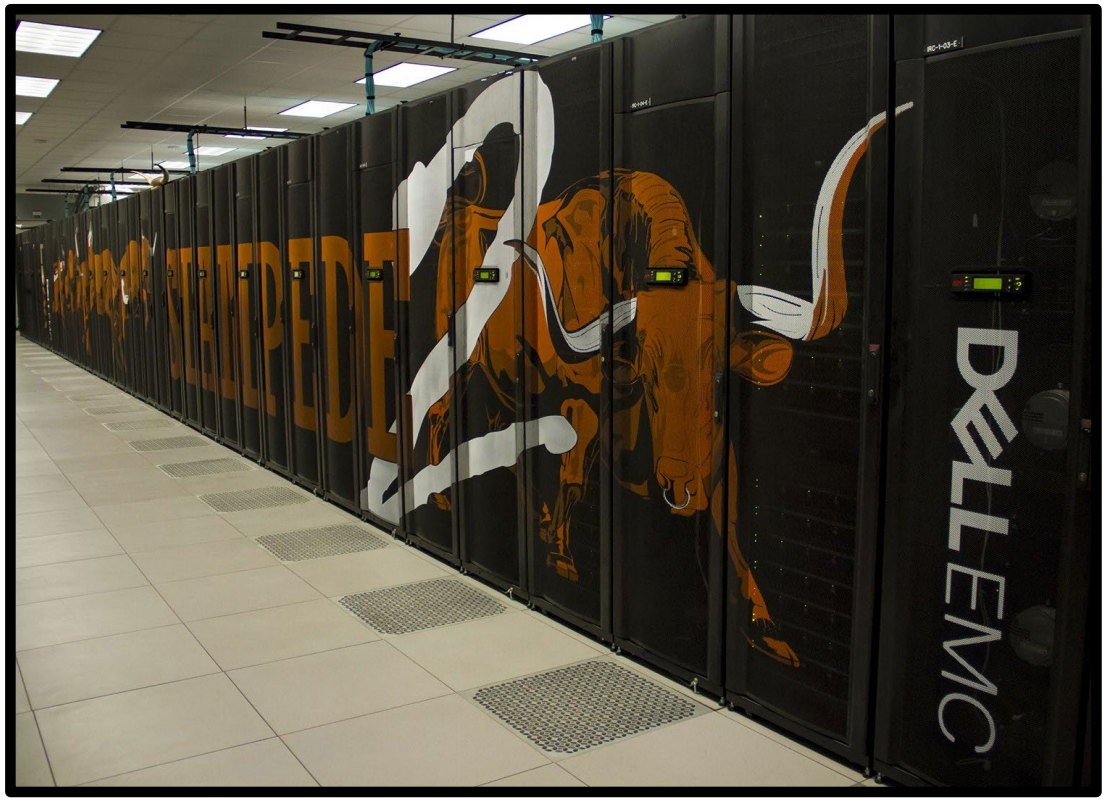
- Hosted by Pittsburgh Supercomputing Center in Monroeville
- Bridges Architecture
- Bridges Virtual Tour

- Large Memory Systems (3TB)
- Many Nvidia GPUs
- Extremely Large Memory Systems (12TB)
- Slurm job scheduler

Computing Options: **Xsede Project (PSC)**

XSEDE

- **Xsede resources**
  - SCSC Comet – 144 GPUs
  - Open Science Grid – Condor - 1000 cores avg available
  - JetStream – IU and TACC – ½ Petaflop



Computing Options: **Cloud Providers**

- **Amazon AWS**
  - https://aws.amazon.com/grants/
  - https://aws.amazon.com/education/awseducate/
- **Google Cloud Compute**
  - https://cloud.google.com/edu/
  - https://lp.google-mkto.com/CloudEduGrants_Faculty.html
- **Microsoft Azure**
  - https://www.microsoft.com/en-us/research/academic-programs/
  - https://azure.microsoft.com/en-us/free/