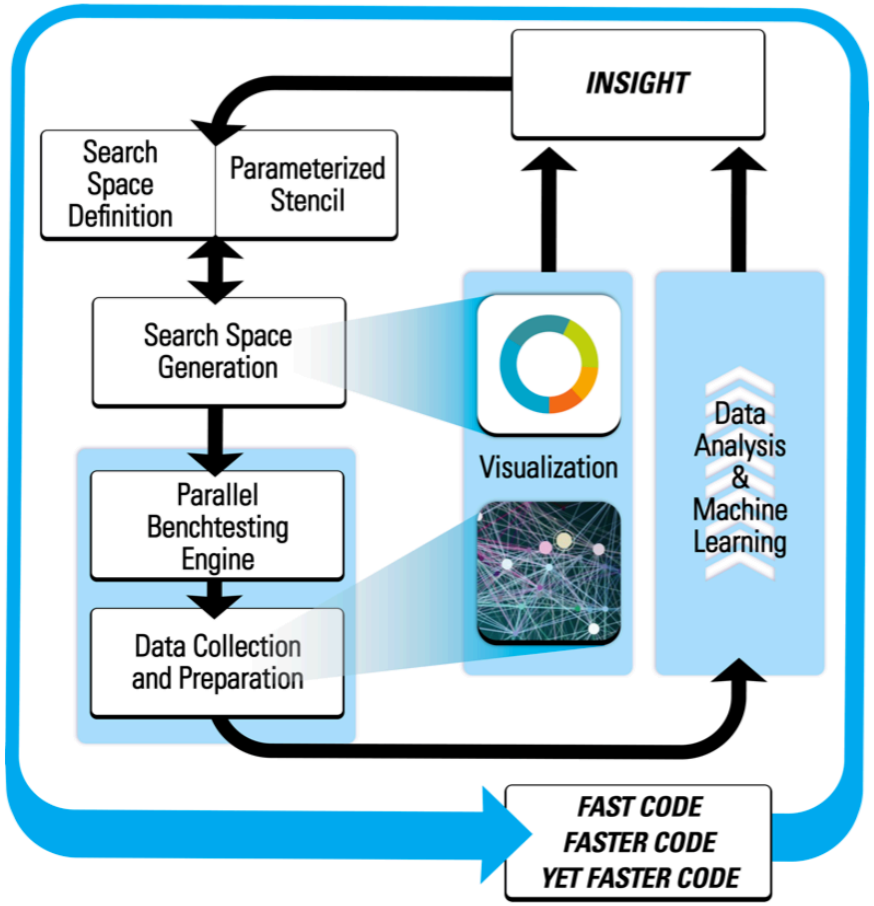
BONSAI: Benchtesting OpeN Software Autotuning Infrastructure

Matthew Bachstein

ICL Lunch Talk

1 Dec 2017

Why BONSAI?



- Accelerators have multiple hardware parameters
- Kernels can have many parameters
- Data can have many different layouts
- Data can have many different characteristics (e.g. Sparse)

Why BONSAI?

- So, can we tune over all these parameters in a framework agnostic way?
- Necessitates a very large parameter search space
  - ~10^5, 10^6, or higher
- Need to do Massive parallel tuning sweeps to achieve reasonable timing

What is BONSAI?

- A full autotuning infrastructure for computational kernels.
  - Language agnostic
- You give us a kernel, we give you:
  - A DSL to specify your parameter space (LANAI)
  - A runtime that will compile/exec/log all of the configurations
  - A visualization/analysis component

LANAI: LANguage for Autotuning Infrastructure

- Python based DSL
- Uses Python syntax and familiar Python functions
- Defines ‘iterators’:
  - Each iterator defines the range of a parameter
  - Can perform arithmetic, boolean comparison, composition, etc.

```
dim_m = range(1, max_threads_dim_x+1)
dim_n = range(1, max_threads_dim_y+1)

@iterator
def blk_m(dim_m):
    return range(dim_m, max_threads_dim_x+1, dim_m)

@iterator
def blk_n(dim_n):
    return range(dim_n, max_threads_dim_y+1, dim_n)

blk_k = range(1, min(max_threads_dim_x, max_threads_dim_y)+1)

@iterator
def dim_vec(arithmetic, precision):
    if precision == DoublePrecision:
        if arithmetic == Real:
            return range(1, 3)
        else:
            return range(1, 2)
    else:
        if arithmetic == Real:
            return range(1, 5, 3)
        else:
            return range(1, 3)
```

LANAI cont.

- LANAI also allows for constraint conditions that prune the overall search space

```
@condition
def over_max_threads(threads_per_block):
    return threads_per_block > max_threads_per_block

@condition
def over_max_regs_per_thread(regs_per_thread):
    return regs_per_thread > max_registers_per_thread

@condition
def over_max_regs_per_block(regs_per_block):
    return regs_per_block > max_regs_per_block

@condition
def over_max_shmem(shmem_per_block):
    return shmem_per_block > max_shared_mem_per_block
```

- Combined -> complex optimization problem
- Combinatorial explosion

LANAI cont.

```
for (dim_m = 1; dim_m < 1025; dim_m += 1)
  for (trans_a = 0; trans_a < 1; trans_a += 1)
    for (trans_b = 0; trans_b < 1; trans_b += 1)
      for (blk_k = 1; blk_k < (min(1024, 1024) + 1); blk_k += 1) {
        if (dim_vec_delta != 0)
          for (dim_vec = dim_vec_start; dim_vec < dim_vec_bound; dim_vec += dim_vec_delta)
            for (dim_n = 1; dim_n < 1025; dim_n += 1)
              for (tex_b = 0; tex_b < 2; tex_b += 1)
                for (tex_a = 0; tex_a < 2; tex_a += 1) {
                  threads_per_block = dim_m * dim_n;
                  if (threads_per_block > max_threads_per_block) continue;
                  if ((threads_per_block % warp_size) != 0) continue;
                  if (vec_mul_delta != 0)
                    for (vec_mul = vec_mul_start; vec_mul < vec_mul_bound; vec_mul += vec_mul_delta) {
                      if (blk_m_delta != 0)
                        for (blk_m = blk_m_start; blk_m < blk_m_bound; blk_m += blk_m_delta) {
                          if (blk_n_delta != 0)
                            for (blk_n = blk_n_start; blk_n < blk_n_bound; blk_n += blk_n_delta) {
                              if (shmem_per_block > max_shared_mem_per_block) continue;
                              if (regs_per_thread > max_registers_per_thread) continue;
                              if (regs_per_block > max_regs_per_block) continue;
                              if ((fmas_per_block / loads_per_block) < min_fmas_per_load) continue;
                              if (max_threads_by_shmem < min_threads_per_multi_processor) continue;
                              if (max_threads_by_regs < min_threads_per_multi_processor) continue;
                              if (dim_n_b_delta != 0)
                              for (dim_n_b = dim_n_b_start; dim_n_b < dim_n_b_bound; dim_n_b += dim_n_b_delta) {
                                  if (dim_n_a_delta != 0)
                                for (dim_n_a =dim_n_a_start; dim_n_a < dim_n_a_bound; dim_n_a +=dim_n_a_delta) {
                                     if (dim_m_a_delta != 0)
                                for (dim_m_a=dim_m_a_start; dim_m_a <dim_m_a_bound; dim_m_a+=dim_m_a_delta){
                                        if (dim_m_b_delta != 0)
                                for (dim_m_b=dim_m_b_start; dim_m_b<dim_m_b_bound; dim_m_b+=dim_m_b_delta){
                                          if ((dim_m_a * dim_n_a) != threads_per_block) continue;
                                          if ((dim_m_b * dim_n_b) != threads_per_block) continue;
                                          if (((((blk_k % (dim_m_b * dim_vec)) != 0) || ((blk_n % dim_n_b) != 0))))
                                            continue;
                                          if (((((blk_m % (dim_m_a * dim_vec)) != 0) || ((blk_k % dim_n_a) != 0))))
                                            continue;
                                          idx += 1;
                                        record(trans_a, trans_b, dim_m, dim_n, blk_m, blk_n, blk_k, dim_vec, vec_mul,
                                               dim_m_a, dim_n_a, dim_m_b, dim_n_b, tex_a, tex_b, idx);
                                        }}}}}}}}}
```

- LANAI compiles to C code
- Accelerated with OpenMP
  - Current work on optimizing the insertion points
- C code generates a CSV file with valid configurations

Runtime

- Parameters are passed as compile time defines
  - E.g. ‘-DBLK_N=32’
- Manager/Worker type process
- Compile step
  - Specify number of compile processes at once
- Execution step
  - Working on multiple gpu support

Runtime cont

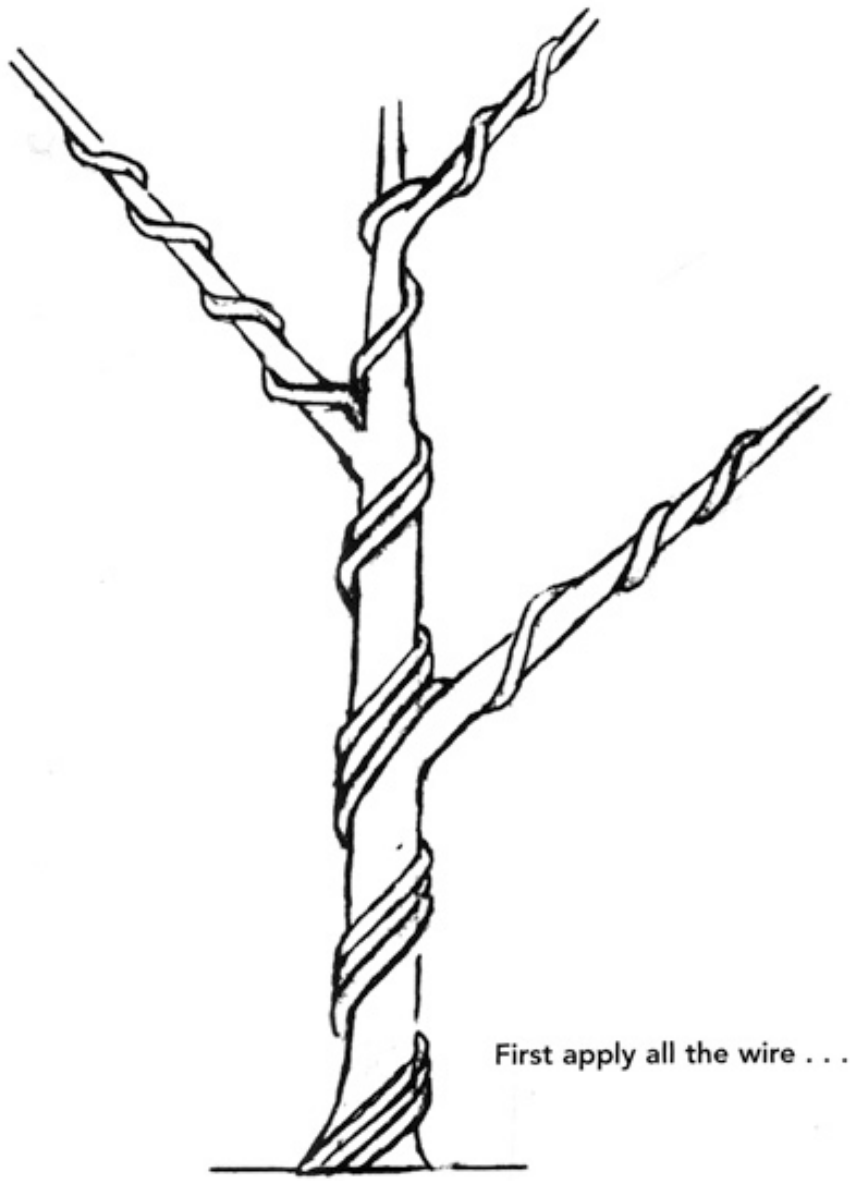
- Modular runtime
  - Single node
  - Cluster
  - Support Cross Compilation (Titan)
- Multiple varieties
  - Staged/full execution
    - Splits the compile/exec stages
  - MPI/No MPI
  - Any combination

Visualization

- Goal: Provide useful analytics about kernel performance
  - Performance/parameter interaction
- Currently Work in Progress
- Some architectural changes are needed first…

Near Term

- More Tutorials
  - OpenCV
  - Packaged Cuda examples
- Optimize the LANAI compiler
  - Find optimal parallelization points
  - Combat the combinatorial explosion
- Data coalesing



Longer Term

- Add SQL logging option
  - Easier to integrate ML tools
- Dynamic job scheduler
- Hardware counter collection wrappers
  - PAPI, CUPTI, etc.
- More robust error catching

Fin

- Questions?

- Required bad joke
- **How do you catch a squirrel?**
- **Climb up a tree, and act like a nut.**