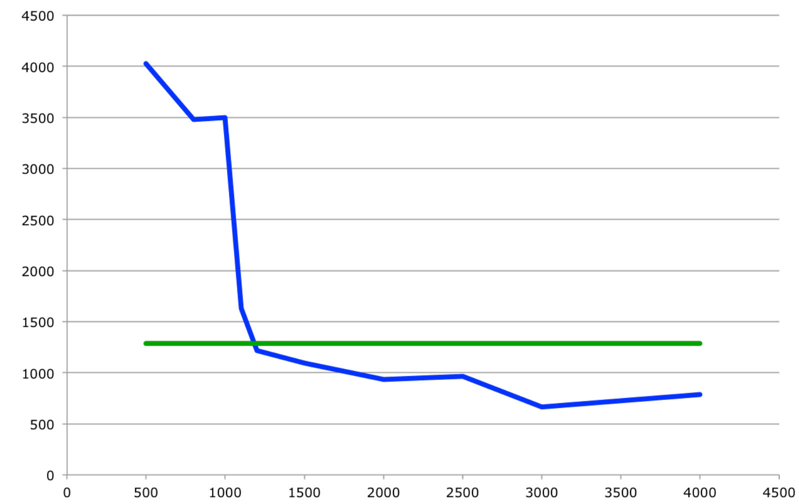
Automating Code Tuning: An Example with Transpose

William Gropp
www.cs.illinois.edu/~wgropp

Transpose Example Review

- do j=1,n
  ```
      do i=1,n
          b(i,j) = a(j,i)
      enddo
  enddo
  ```
- No temporal locality (data used once)
- Spatial locality only if (words/cacheline) * n fits in cache



- Performance plummets when matrices no longer fit in cache

PARALLEL@ILLINOIS

Blocking for cache helps

- do jj=1,n,stridej
```
do jj=1,n,stridej
    do ii=1,n,stridei
        do j=jj,min(n,jj+stridej-1)
            do i=ii,min(n,ii+stridei-1)
                b(i,j) = a(j,i)
```
- Good choices of stridei and stridej can improve performance by a factor of 5 or more
- But what are the choices of stridei and stridej?

But what size of blocks?

- Can we predict from simple performance model
  - Not really, as we'll see from the results
- However, the behavior is not entirely random, so some sampling methods can be effective.

Autotuning

- Really, automate the process of evaluating different parameter (and possibly code) choices to find the “best” (or at least, good enough)
- To use autotuning
  - Need a way to generate code for different parameters

Tools: Code Generation

- Loopy
  - https://documen.tician.de/loopy/
- CHiLL
  - http://ctop.cs.utah.edu/ctop/?page_id=21
- Orio
  - https://brnorris03.github.io/Orio/
- POET
  - http://www.cs.uccs.edu/~qyi/poet/
- And there are others, including ones for special cases, such as the Tensor Contraction Engine

Tools: Autotuning

- OpenTuner
  - ♦ http://opentuner.org/
- Active Harmony
  - ♦ http://www.dyninst.org/harmony/
- Many special purpose environments
  - ♦ Sparse matrices http://bebop.cs.berkeley.edu/oski/

Example: Loopy for transpose

```
trans.f          Thu Mar 24 12:18:58 2016          1

      subroutine init_matrices(matA, matB, matSize)
      implicit none
      integer matSize
      double precision matA(matSize,matSize), matB(matSize,matSize)
      integer i, j

!     Initialize the matrices
      do i=1,matSize
         do j=1,matSize
            matA(i,j) = 1.0 + i
            matB(i,j) = 1.0 + j
         enddo
      enddo
!
      end

      subroutine transp(matA, matB, matSize)
      implicit none
      integer matSize
      double precision matA(matSize,matSize), matB(matSize,matSize)
      integer i, j

      do i=1,matSize
         do j=1,matSize
            matB(i,j) = matA(j,i)
         enddo
      enddo

      end

!$loopy begin
! init_matrices, transp = lp.parse_fortran(SOURCE, FILENAME)
! transp = lp.assume(transp, "matSize > 0")
!
! if 1:
!     bsize_i = 256
!     bsize_j = 256
!     transp = lp.split_iname(transp, "i", bsize_i)
!     transp = lp.split_iname(transp, "j", bsize_j)
!     # if desired
!     # transp = lp.assume(transp, "matSize mod %d = 0" % bsize_i)
!     # transp = lp.assume(transp, "matSize mod %d = 0" % bsize_j)
!     transp = lp.set_loop_priority(transp,
!         "i_outer,j_outer,i_inner,j_inner")
! else:
!     transp = lp.set_loop_priority(transp, "i,j")
!
! RESULT = [init_matrices, transp]
!$loopy end
```

- Code is just the simple (clean, easy to read) code
- Followed by an *annotation* that tells loo.py what transformation to apply

Example Generated Code

trans-16-8.c   Thu Mar 24 12:18:12 2016   1

```
void init_matrices_(double *restrict matA, double *restrict matB, int *const matSize)
{

  for (int j = 0; j <= -1 + *matSize; ++j)
    for (int i = 0; i <= -1 + *matSize; ++i)
    {
      matA[i + *matSize * j] = 2.0 + i;
      matB[i + *matSize * j] = 2.0 + j;
    }
}

void transp_(double const *restrict matA, double *restrict matB, int *const matSize)
{

  for (int i_outer = 0; i_outer <= -1 + ((15 + *matSize) / 16); ++i_outer)
    for (int j_outer = 0; j_outer <= -1 + ((7 + *matSize) / 8); ++j_outer)
      for (int i_inner = 0; i_inner <= 15; ++i_inner)
        if (-1 + -1 * i_inner + -16 * i_outer + *matSize >= 0)
          for (int j_inner = 0; j_inner <= 7; ++j_inner)
            if (-1 + -1 * j_inner + -8 * j_outer + *matSize >= 0)
              matB[i_inner + i_outer * 16 + *matSize * (j_inner + j_outer * 8)] = matA
[j_inner + j_outer * 8 + *matSize * (i_inner + i_outer * 16)];
}
```

- bsize_i = 16
- bsize_8 = 8

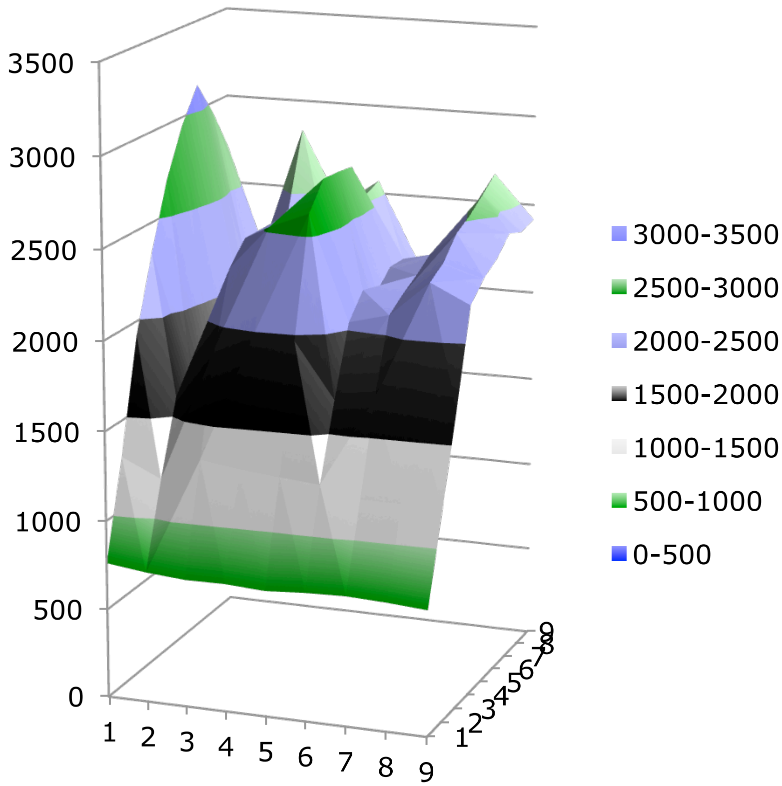
Generating the different parameters

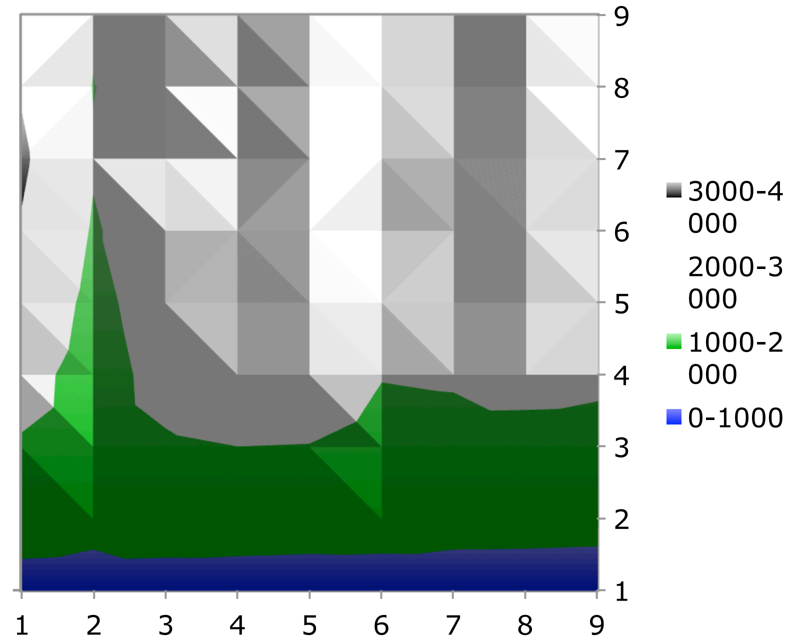
- While one *should* use a tool that can manage the process, a simple approach can work for cases with only a few parameters
- Use a shell script to run through the values of the parameters
- Use sed to create a new source version for each parameter choice
- Invoke the loo.py tool to create each version
- Build executables and run each; collect times
- I did this on Blue Waters (building the sources first on my Macbook)

Example: Testing Transpose

```
#! /bin/sh
# Simple tuning script
bsizes="1 2 4 8 16 32 64 128 256”
source myenv/bin/activate
cp trans.f trans.f.orig
for bsize_i in $bsizes ; do
   for bsize_j in $bsizes ; do
          sed -e "s/bsize_i =.*/bsize_i = $bsize_i/g" \
             -e "s/bsize_j =.*/bsize_j = $bsize_j/g" \
             trans.f.orig > trans.f
          make clean
          make trans
          cp trans.c trans-$bsize_i-$bsize_j.c
          echo "$bsize_i x $bsize_j"
          ./trans
   done
done
```

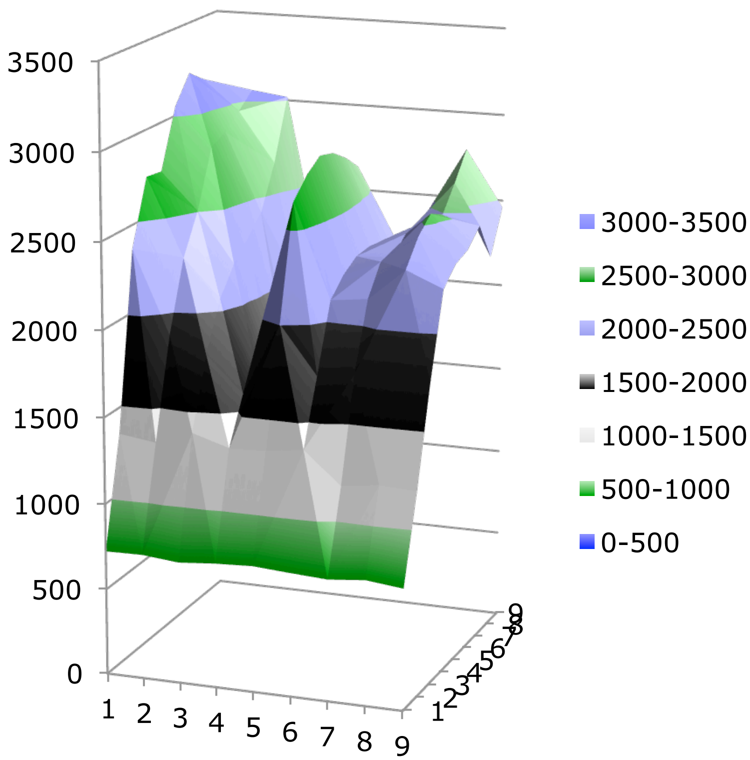
Results: Macbook O1

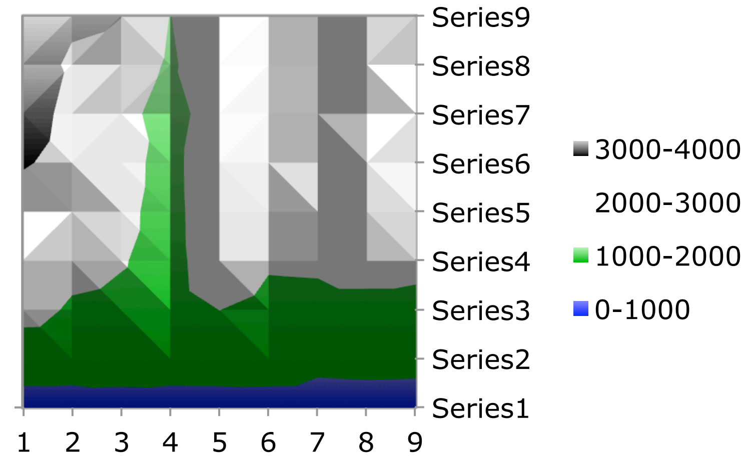




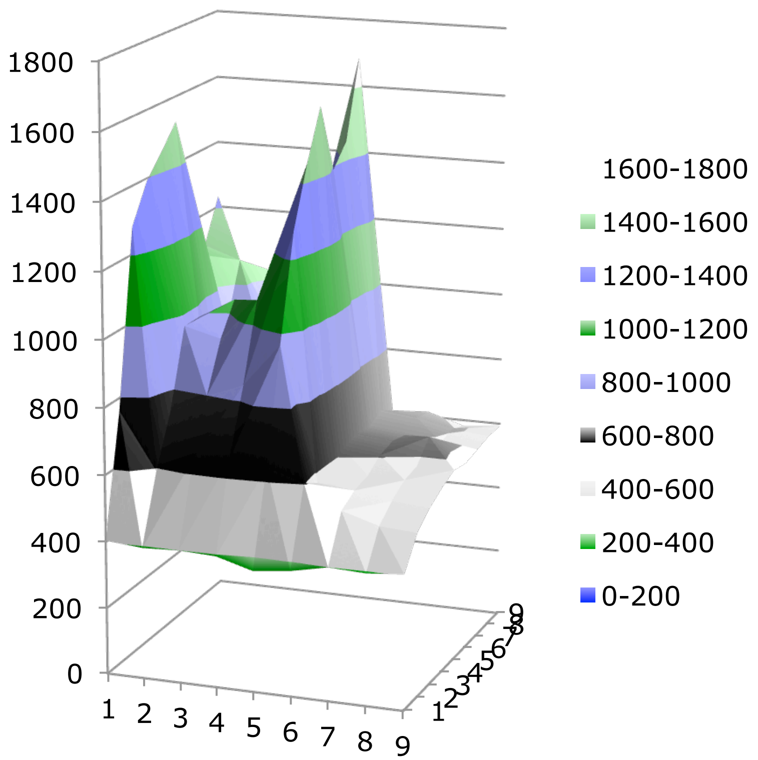
PARALLEL@ILLINOIS

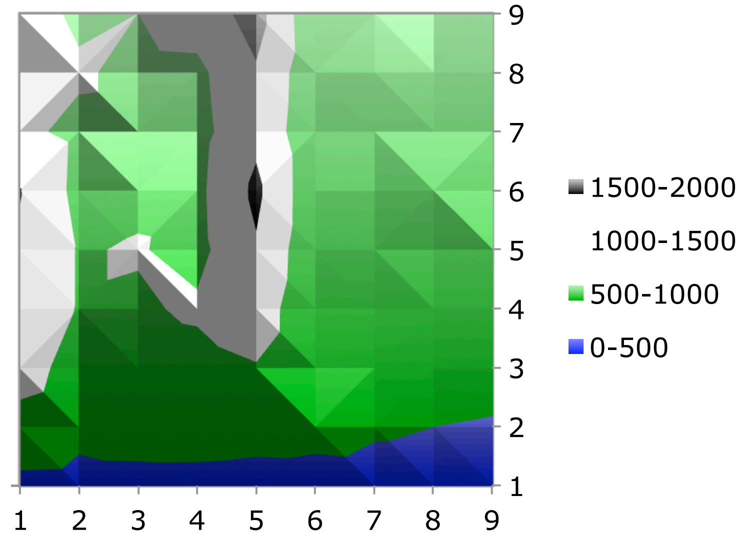
Results: Macbook O3



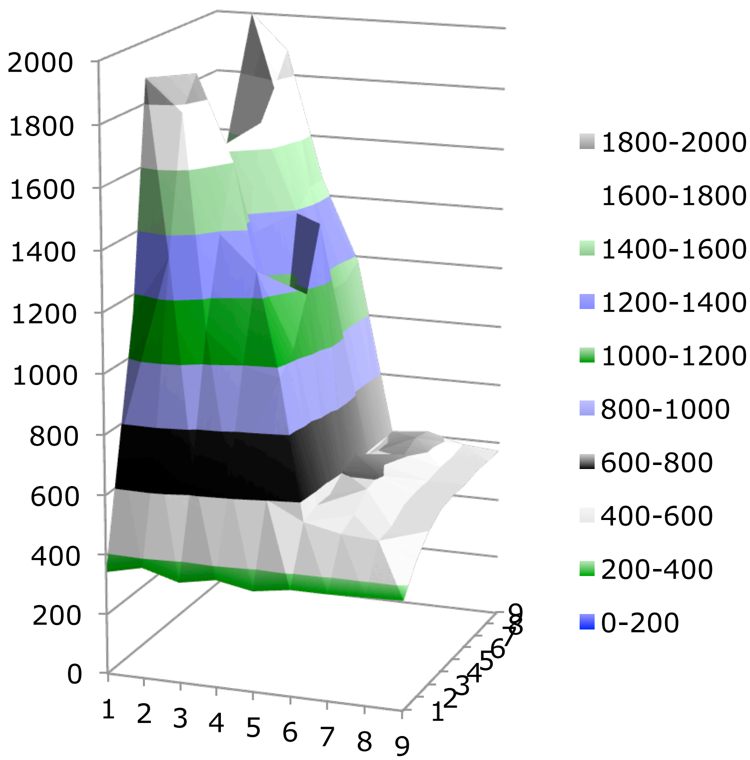


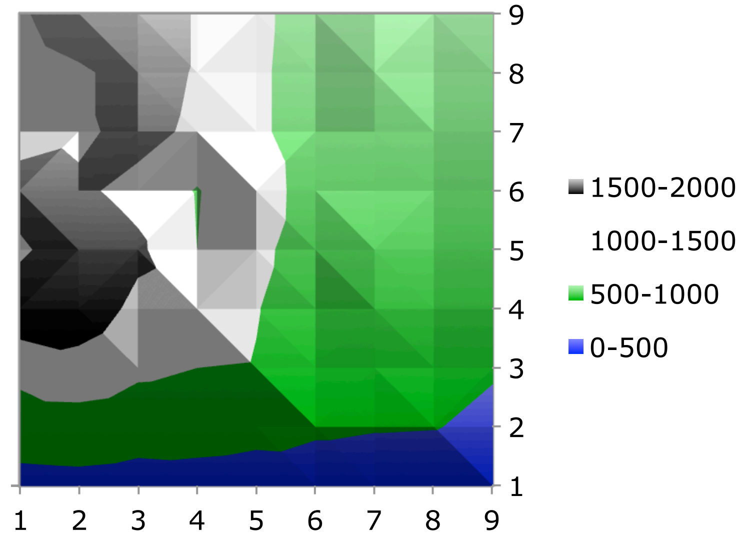
Results: Blue Waters O1





Results: Blue Waters O3





Summary

- Can use many different tools to automate generation of code
- Performance is hard to predict
- Performance depends on system
- Performance depends on compiler optimization, even when generating "optimized" code.
- Performance depends on problem parameters (e.g., size), so may need to be tuned to specific parameters
- For transpose, note that the performance is asymmetric with respect to the block sizes for i and j