• Merge the setup codes and+ stitcher
•Currently, stitch the entire dynamic region at one go: instead just stitch parts as we need them

TRICK IS NOT TO DO COMPLETE ALLOC/SCHED: IMPROVE ON A GOOD ONE
•Use better runtime info on memory accesses to registerize some variables:
on stack-based calculator, speedup increased from 1.7x to 4.1x

•Inlining allows dynamic regions to extend into callee functions... more scope for optimizations and speeudp

•In the distant future, maybe the program will not need to annotate...
••Need to develop a model to do cost-benefit evaluation of dc
••Profiling to identify possible dynamic region and constants

QUESTIONS??

•Simple interpreter, some matrix crunching programs, operating system code, unix utility type program (record sorting)

•Asymptotic speedup is a measure of code quality:
What's the speedup when you run the region a large enough number of times?
Speedups are good(for the compile

•Break-even point is a measure of dynamic compilation overhead:
how many times do you have to re-use to buy back the overhead?
Unacceptably high (e.g. have to scale by a 150x150 matrix)

•Bottleneck: Stitcher has to interpret directives, traverse runtime constant data-structure... insert a fragment of stitcher that uses setup value as soon as it is computed!

WHICH BRINGS US TO WHAT WE PLAN TO DO IN THE FUTURE...

- Red stuff (runtime constant calculations) is folded away
  half the # of instructions, half the # of loads
- Peephole optimizations get rid of divides
- Loop fully unrolled: get rid of branches and index update overhead

SIGNIFICANTLY FEWER, CHEAPER INSTRUCTIONS

WE'VE DONE SOME PRELIMINARY RUNS ON A FEW REPRESENTATIVE PROGRAMS...

Static part: modified optimizing compiler; dynamic part.... lightweight

Need annotations (OVERLAY) identifying dynamic region, dyn consts, loop to be unrolled

Normal code goes through as usual

Static compiler const-prop type dataflow analysis to identify all runtime consts
(red code.... setup code)
(remaining code... templates, with holes for runtime constants)

optimize down to assembler with holes (for rtcs) and markers (for loops)
hears what a template look like... except imagine it's in assembler
(REPLACE LEFT SLIDE WITH TEMPLATE SLIDE)

emit directives to on how to fill holes, process markers

At runtime, when DR is hit, execute setup code; the dynamic compiler INTERPRETS directives to patch holes and unroll loops...

END RESULT OF ALL THIS IS.....

- Cache lookup routine
- Cache configuration fixed early, address changes
- Figure out if tag is in the cache

•Many computations (in red) that involve just the cache config

What we'd like to do....
•Figure out the red computations as being derived from runtime const
•Do these once and optimize the (black) code that uses these values
e.g. unroll loop fully
•In future iterations, execute just the black code....

And actually, we have a system that does just this...

MOVE SLIDE TO LEFT PROJECTOR

It’s doing some compilation at runtime
Some info available only at runtime: specifically, runtime constants
(Variables whose values are fixed at runtime)

Will produce significantly faster code...
...on many applications:
- >cache simulator: cache structure fixed
- >interpreter: code being interpreted fixed... would like a compiled version

But optimizations take time... (cycles to burn for static compiles)
How to keep down the cycles while getting very good code?

Start with very good code... don’t do everything at runtime!
- •Statically optimize and generate code templates
- •”Dynamic compilation” consists of simple actions on templates.

Example to illustrate what I’ve been saying...

Not introduced:

name, topic

Else:

UW dynamic compilation group
(list members?)