POET:
Parameterized Optimizations
For Empirical Tuning

Qing Yi
University of Texas at San Antonio
Extracting high performance through empirical tuning

- Producing high-performance applications is hard
  - It is increasingly hard to predict
    how machines will behave
    how applications will behave
    how compilers will behave
    how the network will behave ...

- The promising solution: empirical tuning
  - Experiment with different transformations to the application
  - Collect performance feedback
  - Adjust transformations and re-experiment
Empirical tuning systems

- Domain-specific libraries
  - Many highly successful
    - ATLAS, PHiPAC, FFTW, SPIRAL...
  - Time consuming: needs to manually orchestrate specialized optimizations
  - Not reusable across different problem domains

- Empirical optimizing compilers
  - Compilers trying to automate the tuning process
  - Hard to incorporate customized optimizations
  - Does not allow human intervention in general

- POET is trying to provide
  - A tool box for programmers to easily build high-performance kernels
  - A communication interface among all components of a tuning system
Empirical tuning approach

- Analysis engine/ Human
  - Understand application and machine, choose optimizations to apply
- Search engine exploits the configuration space
  - Use info from program analysis (encoded in configuration space)
- Transformation engine
  - Transforms application code based on transformation script and search configuration
The POET Language

- Language for expressing result of analysis engine
  - Parameterized code transformation scripts
  - Parameterized configuration space

- Interpreted by search engine and transformation engine
  - Configuration space interpreted by independent search engine
    - Sequence of parameters, constraints on parameter values
  - Transformations applied by transformation engine
    - Configuration determined by search engine
Flexibility, Modularity and Efficiency

- Portability --- applications can be shipped in POET representation
  - Tuned by independent search and transformation engines on different platforms

- Efficiency --- both transformation and search engines are light-weight
  - Heavy weight analysis optimizations done only once in analysis and optimization engine
  - Result parameterized to be tuned many times on different platforms

- Flexibility --- analysis engine and transformation/search engine can reside on different machines
  - Analysis engine not involved in the tuning process
  - Analysis and tuning research is separate and independent
  - Different optimizations can be combined through an external common language
POET: Describing Structure of Input computation

Example: code templates for C in POET

```xml
<code Nest pars=(loop, body) >
  @loop@ {
    @body@
  }
</code>

<code Sequence pars=(s1,s2) >
  @s1@
  @s2@
</code>

<code Loop pars=(i,start,stop,step) >
  @init=(if start="" then "" else (i ":=" start ));
  test=(if stop="" then "" else (i ":=" stop ));
  incr=(if step="" then "" else (i ":=" step ));
  @for (@init@; @test@; @incr@)
</code>

- POET: scripting language to be embedded in arbitrary languages
  - Syntax of source language described in a collection of code templates
  - Support strings, integers, lists, tuples, tables, AST based on code templates
  - Support loops, conditionals, recursion
  - Predefined library of code transformation routines

- Analysis engine
  - Decompose application into code templates
  - Specify transformations
POET: Describing Structure of Input computation

Example: the input specification for dgemm

```c
//@; BEGIN(gemm)
void ATL_USERMM(const int M, const int N, const int K,
    const double alpha, const double *A, const int lda,
    const double *B, const int ldb, const double beta,
    double *C, const int ldc)             //@=>_:Exp
{
    int i, j, l;                                               //@=>gemmDecl:Stmt; BEGIN(gemmBody)
    for (j = 0; j < N; j += 1)                     //@ =>loopJ:Loop BEGIN(nest3)
    {                                                          //@; BEGIN(body3)
        for (i = 0; i < M; i += 1)                   //@=>loopI:Loop BEGIN(nest2)
            {                                                       //@;BEGIN(body2) BEGIN(parse)
                C[j*ldc+i] = beta * C[j*ldc+i];    //@END(parse) =>_:Stmt
                for (l = 0; l < K; l +=1)                 //@=>loopL:Loop BEGIN(nest1)
                    {                                                     //@;BEGIN(parse)
                        C[j*ldc+i] += alpha * A[i*lda+l] * B[j*ldb+l];  //@END(parse) =>stmt1:Stmt
                    }                                                    //@END(nest1:Nest) END(body2:Sequence)
            }                                                      //@END(nest2:Nest) END(body3:Nest)
    }                                  //@END(nest3:Nest) END(gemmBody:Nest) END(_:Sequence)
}                                                           //@END(gemm:Function)
```
Define transformations in POET

- POET is a dynamic functional language designed for the ease of writing code transformations
  - Supports pattern matching, code traversal, replacement, and duplication
  - Support control flows and recursion
  - support auto tracing of code fragments going through transformations

- Libraries to support most existing code transformations known to be important
POET: What Analysis Engine Needs to Write?

<parameter SSELEN=16, SSEN0=16 />
<parameter mu=6, nu=1, ku=36, NB=36, MB=36, KB = 36, PF=1 />

<trace nest3,loopJ, body3, nest2, loopI, body2,  
  nest1, loopL, stmt1, gemm, gemmDecl, gemmBody/> 

<define Specialize DELAY {
  if (SP) {
    REPLACE("N",NB, loopJ); REPLACE("M",MB, loopI); REPLACE("K",KB, loopL);
    REPLACE("lda", MB, gemmBody); REPLACE("ldb", NB, gemmBody);
    if (alpha == 0) { REBUILD(REPLACE("alpha", 1, gemmBody) }  
  } } />

<define nest3_UnrollJam DELAY {
  if (mu > 1 || nu > 1) {
    UnrollJam[factor=(nu mu)](nest1, nest3, gemmBody);
  } } />

<define nest1_Unroll DELAY {
  if (ku > 1) {
    UnrollLoops[factor=ku](stmt1, nest1, body2);
  } } />

......
POET: Applying Transformations

Writing a POET script
- Define transformation parameters
- Define the input computation
- Define tracing variables
- Define each transformation independently
- Apply transformations and output

......

<output dgemm_kernel.c (  
    TRACE gemm;  
    APPLY Specialize;  
    APPLY A_ScalarRepl;  
    APPLY nest3_UnrollJam;  
    APPLY B_ScalarRepl;  
    APPLY C_ScalarRepl;  
    APPLY array_ToPtrRef;  
    APPLY Abuf_SplitStmt;  
    APPLY body2_Vectorize;  
    APPLY array_FiniteDiff;  
    APPLY body2_Prefetch;  
    APPLY nest1_Unroll;  
    gemm  
) />
Empirical tuning using POET

Compared to empirical tuning using a source-to-source loop optimizer
Collaborated work with You, Seymour (UTK), Quinlan and Vuduc (LLNL)
Automatic production of ATLAS library kernels

Library kernels are
- Computational intensive routines
- Invoked by higher-level procedures
- Assumptions can be made about pre-applied optimizations, e.g., specialization on matrix sizes, array copy

Using POET to automatically produce ATLAS library kernels
- Define the input computation in C with POET annotations
  - Define global variables to trace fragments to be optimized
- Define a collection of potential optimizations independently
  - Invoke the optimizations and output
- Parameterization: precision of kernel, how to apply optimizations

Automatically produced ATLAS kernels: gemm, gemv, ger
- Encoded the relevant ATLAS optimizations in POET
  - Specialization, unroll-and-jam, loop unroll, scalar repl, strength reduction, SSE vectorization
## Experimental Design

<table>
<thead>
<tr>
<th>Platform</th>
<th>Compiler</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.66GHz Core2Duo</td>
<td>icc 9.1</td>
<td>-xP -msse3 -O3 -mp1 -fomit-frame-pointer</td>
</tr>
<tr>
<td></td>
<td>gcc 4.0.1</td>
<td>-mfpmath=sse -msse3 -O2 -m64 -fomit-frame-pointer</td>
</tr>
<tr>
<td>2.2GHz Athlon 64 X2</td>
<td>gcc 4.2.0</td>
<td>-mfpmath=387 -falign-loops=4 -fomit-frame-pointer</td>
</tr>
</tbody>
</table>

### Peak performance of machines

<table>
<thead>
<tr>
<th>Precision</th>
<th>Core2Duo Peak MFLOPS</th>
<th>Athlon-64 X2 Peak MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>scalar Unit</td>
<td>vector Unit</td>
</tr>
<tr>
<td>single</td>
<td>5320</td>
<td>21280</td>
</tr>
<tr>
<td>double</td>
<td>5320</td>
<td>10640</td>
</tr>
</tbody>
</table>

ATLAS version 3.7.30; Collaborated work with Clint Whaley
## Result for Level-3 BLAS

<table>
<thead>
<tr>
<th>MFLOP/%PEAK</th>
<th>Core2Duo MFLOPS</th>
<th>Athlon-64 MFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc +ref</td>
<td>icc +ref</td>
<td>ATLAS gen</td>
</tr>
<tr>
<td>sgemmK</td>
<td>571/2.7%</td>
<td>6226/29.3%</td>
</tr>
<tr>
<td>dgemmK</td>
<td>649/6.1%</td>
<td>3808/35.8%</td>
</tr>
</tbody>
</table>

PEAK: using vector (SSE) units
## Result for Level-2 BLAS

<table>
<thead>
<tr>
<th></th>
<th>Core2Duo MFLOPS/ %PEAK</th>
<th>Athlon-64 MFLOPS/ %PEAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>sgerK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc+ref</td>
<td>1230/5.7%</td>
<td>639/7.3%</td>
</tr>
<tr>
<td>icc +ref</td>
<td>2927/13.7%</td>
<td>1005/11.4%</td>
</tr>
<tr>
<td>ATLAS full</td>
<td>3751/17.6%</td>
<td>1005/11.4%</td>
</tr>
<tr>
<td>POET +ref</td>
<td>3400/15.9%</td>
<td>1005/11.4%</td>
</tr>
<tr>
<td>dgerK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc+ref</td>
<td>411/9.3%</td>
<td>518/11.8%</td>
</tr>
<tr>
<td>icc +ref</td>
<td>438/4.1%</td>
<td>500/11.4%</td>
</tr>
<tr>
<td>ATLAS full</td>
<td>519/4.9%</td>
<td>500/11.4%</td>
</tr>
<tr>
<td>POET +ref</td>
<td>401/11.8%</td>
<td>500/11.4%</td>
</tr>
<tr>
<td>sgemvTK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc+ref</td>
<td>1752/8.2%</td>
<td>1389/15.8%</td>
</tr>
<tr>
<td>icc +ref</td>
<td>1826/8.6%</td>
<td>2056/23.4%</td>
</tr>
<tr>
<td>ATLAS full</td>
<td>2171/10.2%</td>
<td>2056/23.4%</td>
</tr>
<tr>
<td>POET +ref</td>
<td>835/9.5%</td>
<td>2056/23.4%</td>
</tr>
<tr>
<td>dgemvTK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc+ref</td>
<td>835/7.8%</td>
<td>739/16.8%</td>
</tr>
<tr>
<td>icc +ref</td>
<td>574/5.4%</td>
<td>1049/23.8%</td>
</tr>
<tr>
<td>ATLAS full</td>
<td>1079/10.1%</td>
<td>1049/23.8%</td>
</tr>
<tr>
<td>POET +ref</td>
<td>579/13.2%</td>
<td>1049/23.8%</td>
</tr>
<tr>
<td>sgemvNK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc+ref</td>
<td>1838/8.6%</td>
<td>1185/13.5%</td>
</tr>
<tr>
<td>icc +ref</td>
<td>2097/9.8%</td>
<td>1986/22.6%</td>
</tr>
<tr>
<td>ATLAS full</td>
<td>2097/9.8%</td>
<td>1986/22.6%</td>
</tr>
<tr>
<td>POET +ref</td>
<td>529/6.0%</td>
<td>1986/22.6%</td>
</tr>
<tr>
<td>dgemvNK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc+ref</td>
<td>939/8.8%</td>
<td>799/18.2%</td>
</tr>
<tr>
<td>icc +ref</td>
<td>1069/10.0%</td>
<td>902/20.5%</td>
</tr>
<tr>
<td>ATLAS full</td>
<td>1069/10.0%</td>
<td>902/20.5%</td>
</tr>
<tr>
<td>POET +ref</td>
<td>408/9.3%</td>
<td>902/20.5%</td>
</tr>
</tbody>
</table>
Improvement For LAPACK (Core2Duo)
Improvement For LAPACK (Athlon X2)
Summary

- **POET targets**
  - General-purpose compiler transformations
    - Separate transformation from program analysis
      - They are separated within compilers anyway
      - Each transformation: replace an AST fragment with a new one
  - Parameterized interface for applying transformations
    - Loop fusion, interchange, blocking, unroll-and-jam, unrolling, scalar replacement, array copy, SIMD vec., prefetching, strength reduction ...

- **Domain-specific transformations**
  - Equivalent algorithms that might return different results
    - precision of floating point numbers, error analysis
  - Using POET to define customized transformations
    - Write transformation routines
    - Can operate on high-level concepts
POET in Empirical Tuning

- Compilers/analyzers => POET transformation engine
  - Compilers/analyzers
    - Discover performance-critical routines; apply outlining; insert POET computation annotations
    - Discover profitable optimizations; produce parameterized transformation scripts (invocations to POET libraries)
  - Export program analysis results to POET
    - Dependence constraints, insight about programs
    - Information useful to the empirical search engine
  - POET transformation engine
    - Interpret the POET scripts; apply transformations; output result

- POET transformation engine => Empirical tuning
  - Search driver decides configuration of transformation parameters
  - POET transformation engine acts as the code generator