Generic Code Optimization

Jack Dongarra, Shirley Moore, Keith Seymour, and Haihang You

LACSI Symposium
Automatic Tuning of Whole Applications Workshop
October 11, 2005
Generic Code Optimization

• Take generic code segments and perform optimizations via experiments (similar to ATLAS)

• Collaboration with ROSE project (source-to-source code transformation / optimization) at Lawrence Livermore National Laboratory
  – Daniel Quinlan and Qing Yi
GCO

- A source-to-source transformation approach to optimize arbitrary code, especially loop nests in the code.
  - Machine parameters detection
  - Source to source code generation
  - Testing driver generation
  - Empirical search engine
GCO Framework

- code
- front end
- IR
- Loop Analyzer
- CG
- Search Engine
- Driver generator
- testing driver
- tuning parameters
- info of tuning parameters

+
Simplex Method

- Simplex method is a non-derivative direct search method for optimal value
- N+1 points of N dimension search space make up a simplex
- Basic Operations: reflection, expansion, contraction, and shrink.
Basic idea of Simplex Method in 2D:

- To find maximum value of \( f(x) \) in a 2-dim search space
- The simplex consists \( X_1, X_2, X_3 \). Suppose \( f(X_1) \leq f(X_2) \leq f(X_3) \)
- In each step, we can find \( X_c \) which is the centroid of \( X_2 \) and \( X_3 \), replace \( X_1 \) with \( X_r \) which is the reflection of \( X_1 \) through \( X_c \).
DGEMM ATLAS Search Space

8 dimensional space for search

ATLAS does orthogonal searching

Represents 1 M search points!!

NB: Cache Blocking  LAT: FP unit latency
MU NU: Register Blocking  KU: unrolling
FFTCH: determine prefetch of matrix C into registers
IFTCH NFTCH: determine the interleaving of loads with computation

simplex32  LAT NB MU NU KU FFTCH IFTCH NFTCH
upper bound  16  32 16 16 32 1 16 16
lower bound  1  16 1 1 1 0 2 1
Comparison of performance of DGEMM generated with ATLAS and Simplex search
Comparison of performance of DGEMM generated with ATLAS and Simplex search
Comparison of performance of DGEMM generated with ATLAS and Simplex search
Comparison of performance of DGEMM generated with ATLAS and Simplex search
Comparison of performance of DGEMM on 1000x1000 matrix generated with ATLAS and Simplex search
Comparison of parameters’ search time
ATLAS and Simplex search

![Bar chart showing comparison of search times for ATLAS and Simplex for different processors: itanium2, power4, pentium4, and sparc. The graph indicates that itanium2 has significantly higher search times compared to the other processors.](image)
Comparison of performance of DGEMM generated with ATLAS and Parallel GA (GridSolve)
Comparison of parameters’ search time
ATLAS and Parallel GA (GridSolve)
Code Generation

- Collaboration with ROSE project (source-to-source code transformation/optimization) at Lawrence Livermore National Laboratory

- LoopProcessor -bk3 4 -unroll 4 ./dgemv.c
Testing Driver Generation

- Testing driver initializes variables and collects performance data.

- Wallclock time or Hardware counter data
int min(int ,int);

void dgemv(int M, int N, double alpha, double* A, double* B, double* C)
{
    int i, j;

    /* matrices are stored in column major */
    for (i = 0; i < M; ++i) {
        for (j = 0; j < N; ++j) {
            C[i] += alpha * A[j*M + i] * B[j];
        }
    }
}
Comparison of performance of DGEMV generated with ATLAS and Simplex search with ROSE
Comparison of performance of DGEMV generated with ATLAS and Simplex search with ROSE
Comparison of performance of DGEMV generated with ATLAS and Simplex search with ROSE
Comparison of performance of DGEMV generated with ATLAS and Simplex search with ROSE
void dgemv(int M, int N, double alpha, double * A, double * B, double * C)
{
    int _var_1;
    int _var_0;
    int i;
    int j;
    int ub1, ub2;

    for (_var_1 = 0; _var_1 <= -1 + M; _var_1 += 113)
    {
        ub1 = rosemin((-8 + M),(_var_1 + 105));
        for (_var_0 = 0; _var_0 <= -1 + N; _var_0 += 113)
        {
            ub2 = rosemin((N + -6),(_var_0 + 107));
            for (i = _var_1; i <= ub1; i += 8) {
                for (j = _var_0; j <= ub2; j += 6) {
                    register double bjp0, bjp1, bjp2, bjp3, bjp4, bjp5;
                    register double cip0, cip1, cip2, cip3, cip4, cip5, cip6, cip7;
                    register int t0, t1, t2, t3, t4, t5;
                    
                    t0 = j * M + i;
                    t1 = (1 + j) * M + i;
                    t2 = (2 + j) * M + i;
                    t3 = (3 + j) * M + i;
                    t4 = (4 + j) * M + i;
                    t5 = (5 + j) * M + i;
                    cip0 = C[i];
                    cip1 = C[i + 1];
                    ...
                }
            }
        }
    }
    for (; j <= rosemin(N-1,_var_0+112); j++) {
        C[i] += (alpha * A[(j * M + i)]) * B[j];
        C[i+1] += (alpha * A[(j * M + i+1)]) * B[j];
        ...}
    for (; i <= rosemin((M-1),(_var_1 + 112)); i++) {
        for (j = _var_0; j <= rosemin((N + -1),(_var_0 + 112)); j++) {
            C[i] += (alpha * A[(j * M + i)]) * B[j];
        }
    }
}

cip2 = C[i + 2];
cip3 = C[i + 3];
cip4 = C[i + 4];
cip5 = C[i + 5];
cip6 = C[i + 6];
cip7 = C[i + 7];
bjp0 = B[j];
bjp1 = B[j+1];
bjp2 = B[j+2];
bjp3 = B[j+3];
bjp4 = B[j+4];
bjp5 = B[j+5];
cip0 += (A[t0]) * bjp0;
........
C[i+6] = cip6;
C[i+7] = cip7;
}