Optimizing Floating Point Calculations, II

Michael A. Saum

msaum@math.utk.edu

Department of Mathematics

University of Tennessee, Knoxville

Overview

- Algorithms for Cache.
- Compiler choices and options.
- High Performance libraries.
- Performance Monitoring.

Cache Misses

- Compulsory cache misses. These misses occur when the cache line has to be brought into the cache when first accessing it. Unavoidable.
- Capacity cache misses. These misses are related to the limited size of the cache preventing all the necessary data to be in cache simultaneously. New data brought into cache may have to overwrite older entries.
- Conflict cache misses. These misses occur in set associative caches due to the fact that each line in memory can only go into selected areas in cache. This makes the effective cache size appear smaller than the physical cache size.

Data Access Transformations

- Various techniques can be used to reduce capacity misses for a particular level of cache:
  - Loop Interchange/Reordering
  - Loop Fusion
  - Loop Blocking/Tiling
  - Prefetching
- *Stride* is the distance (measured in words) between two memory locations consecutively accessed by a program. Unit stride is best for cache locality.
- Most cache optimizations target $L2$ cache, although there are techniques which target register and $L1$ cache.

Loop Interchange/Reordering

- Works when order of loop execution is not important.
- FORTRAN accesses storage by column, C by row.

```
// Original Loop nest:
for $i = 1$ to $n$ do
   for $j = 1$ to $n$ do
      $sum += a[i, j]$
   end for
end for

// Interchanged Loop nest:
for $j = 1$ to $n$ do
   for $i = 1$ to $n$ do
      $sum += a[i, j]$
   end for
end for
```

Loop Fusion

- Takes two adjacent loops that have same iteration space traversal and combines into single loop.

```
// Original code:
for $i = 1$ to $n$ do
    $b[i] = a[i] + 1.0$
end for
for $i = 1$ to $n$ do
    $c[i] = b[i] + 4.0$
end for

// After Loop Fusion:
for $i = 1$ to $n$ do
    $b[i] = a[i] + 1.0$
    $c[i] = b[i] + 4.0$
end for
```

Loop Blocking/Tiling

- Adds additional depth to a loop nest.
- Improves data locality in cache.
- 1D or Line Blocking is useful when multiple long vectors being operated on in same loop.
- 2D or Square Blocking is useful when large matrix-vector or matrix-matrix operations occur in same loop.
- Assume in following examples that n_s is a value which takes into account how much data can fit in cache at the same time.

Line Blocking

```
// Original code:
for $i = 1$ to $n$ do
    $b[i] = a[i] + c[i]$
    $OtherCalcs$
end for

// Line Blocked Code:
for $i = 1$ to $n$ by $n_s$ do
    for $ii = i$ to $\min(n, i + n_s - 1)$ do
        $b[ii] = a[ii] + c[ii]$
        $OtherCalcs$
    end for
end for
```

Square Blocking

```
// Original code:
for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do
        $a[i, j] = b[j, i]$
    end for
end for

// Square Blocked Code:
for $i = 1$ to $n$ by $n_s$ do
    for $j = 1$ to $n$ by $n_s$ do
        for $ii = i$ to $\min(n, i + n_s - 1)$ do
            for $jj = j$ to $\min(n, j + n_s - 1)$ do
                $a[ii, jj] = b[jj, ii]$
            end for
        end for
    end for
end for
```

Data Prefetching

- Frequently the CPU is not fed fast enough with data from memory., which can be a bottleneck resulting in the CPU waiting for data to process.
- Data prefetching can be used to request data be brought into cache before it is to be used.
- Successful technique when the data stream can be predicted correctly.
- Successful when it doesn’t interfere with still active references in cache.
- Pentium 4 has assembler commands which can be inserted into code.
- Much easier to do in C than FORTRAN.

Loop Unrolling

- Use temporary variables to reduce the number of loops executed by manually performing certain operations.
- Usually used to increase instruction level parallelism, blocking for registers.

```
// Original code:
$tt = 0$
**for** $i = 1$ **to** $n$ **do**
    $tt = tt + a[i] * a[i]$
**end for**

// Loop Unrolling:
// Works correctly only if $n$ is divisible by 4:
$tt = 0$
**for** $i = 1$ **to** $n$ **by** $4$ **do**
    $tt = tt + a[i] * a[i] + a[i+1] * a[i+1] + a[i+2] * a[i+2] + a[i+3] * a[i+3]$
**end for**
```

Data Layout Transformations

- Aimed to address cache conflict misses and improve spatial locality.
- Modify how data structures or variables are laid out in memory.
- Techniques include:
  - Changing base addresses of variables,
  - Modifying array sizes (Padding),
  - Transposing array dimensions,
  - Merging of arrays,
  - Data copying, non-contiguous to contiguous.
- Techniques usually applied at compile time, although some optimizations can also be applied at run time.

Compiler Choices

- GNU compiler collection
  - Available on all Linux machines
  - `gcc` - `C`
  - `g++` - `C++`
  - `g77` - `FORTRAN 77`
- Intel Compiler Collection
  - Available on `agnesi`, will be made available on all desktop machines in the department Spring/Summer 2006.
  - `icc` - `C/C++`
  - `ifc` - `FORTRAN 77/90/95`

Basic Compiler Optimizations

- Compilers will perform a wide variety of optimizations, if asked.
  - Common subexpression elimination,
  - Strength reduction - replacement of arithmetic expression by equivalent expression which runs faster,
  - Loop invariant code moved outside of loop,
  - Constant value propagation/evaluation,
  - induction variable simplification, mainly used in calculation of array addresses,
  - register allocation and instruction scheduling.

Compiler Options

- Luckily, not all code optimizations need to be done manually. Passing the right options to a compiler will ask the compiler to attempt to perform optimizations.
- Note that there is no guaranty that aggressively optimizing a code will improve the run time of the code. Sometimes it runs slower. User beware!
- There are some common options to both the GNU and Intel collections:
- Detailed descriptions of all options are contained in the `man` pages for the compiler being used.

Compiler Options, contd.

- Optimization
  - `-O0` - No optimization (Useful for debugging),
  - `-O1` - Minimal optimizations attempted,
  - `-O2` - Medium optimizations peformed,
  - `-O3` - Aggressive optimizations.
- Debugging
  - `-g` - Include debugging symbol table in executeable, useful for debugging but does not cause program performance degradation.
  - `-pg` - Produce profiling information to be processed later.
  - `-Wall` - Produce output for all warnings during compilation.

Useful GNU Specific Options

- `-mcpu=pentium4` - tune to the pentium4 everything about the generated code.
- `-march=pentium4` - generate instructions for pentium4
- `-mfpmath=sse` - use scaler floating point instructions present in SSE instruction set.
- `-msse` - enable use of SSE built-in functions.
- `-msse2` - enable use of SSE2 built-in fucntions.
- `-malign-double` - aligns doubles on a two-word boundary.
- `-funroll-loops` - unroll loops whose number of iterations can be determined at compile time or upon entry to the loop.

Useful Intel Specific Options

- `-tpp7` - Optimize non-exclusively for pentium4 processor (same as `-mcpu=pentium4` GNU option)
- `-xWN` - Optimize exclusively for pentium4 processor (same as `-march=pentium4` GNU option)
- `-axWN` - Generates code for pentium4 processor and generic IA-32 processor, which one is executed determined at run time.
- `-ip` - Enable single file function inlining.
- `-ipo` - Enable multi-file function inlining.
- The Intel compilers have many more options which can be used to fine tune an application.

BLAS/LAPACK/ATLAS

- Basic Linear Algebra Subroutines (BLAS)
  - Level 1 - vector operations (dot product, axpy)
  - Level 2 - matrix - vector operations
  - Level 3 - matrix - matrix operations
- Built on top of BLAS is LAPACK which provides a set of routines to do factorization and solves among other things.
- Automatically Tuned Linear Algebra System (ATLAS) when installed on a system empirically determines and produces a set of BLAS routines (some LAPACK also) which are optimized for that particular system.

Intel MKL

- Intel Math Kernel Libraries (MKL)
- BLAS, Sparse BLAS, and LAPACK routines
- ScaLAPACK routines
- Sparse Solver
- Vector Mathematical Library (VML) replacement for scaler transcendental functions
- Vector Statistical Distribution functions
- Discrete Fourier and Fast Fourier Transform Routines
- One can in general link GNU compiled libraries with Intel compiled libraries, although care should be taken when doing this.

GNU Scientific Library (GSL)

- While not necessarily optimized, GSL contains many numerical routines which one might want to have available but would not want to code.
- Think `C/C++` version of *Abramowitz and Stegun*, *Handbook of Mathematical Functions* on steroids.
- The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total.
- Why code when the code already exists?

Performance Monitoring

- The oldest and the best way to evaluate a code's performance is to write information (timing, counts) at various locations in the code.
- With the GNU compiler system, compiling with the `-pg` flag will insert appropiate code to obtain timing information on the program when run.
- When run the program will create a file called `gmon.out`.
- This file can then be processed by the program `gprof` to obtain function (or even line!) profiling information such as total number of function calls and total time spent in each routine.

PAPI

- Performance API (PAPI) is a set of routines which allows one access to hardware performance counters available on most modern microprocessors.
- On Intel Pentium4, there is a limited set of hardware counters available.
- The maximal *useful* set which can currently be obtained for Pentium4 apps: TOT_CYC, FP_OPS, TOT_IIS, TOT_INS, RES_STL, L1_DCM, L2_TCM, TLB_DM, L2_LDM
- OP counts are somewhat suspect when mixing SIMD/SSE2 vector FP operations with scaler FP operations.

PAPI, contd.

- When installed on IA-32 Linux based systems, the Linux OS kernel must be patched. This requires root access to the OS.
- PAPI can also obtain timing information with very accurate timers, and can produce timing characteristics for wall clock and cpu time.
- Excellent tool for identifying cache behavior with minimal monitoring overhead.

Data Post Processing

- For large and complex programs, one can write data to intermediate files which can then be processed and analyzed by other programs.
- For most data post processing, I write files in some sort of column oriented format.
- For example, at each time step I may write all calculations associated with time t to a file to be processed later.
- There are many other plotting and visualization programs out there. MATLAB is pretty good for quick and dirty calculations/plots.

Post Processing, contd.

I use four basic programs to analyze data.

- `gnuplot` - quick and dirty UNIX plotting (2D,3D) program
- `R` - SAS and MATLAB on steroids.
- `plotmtv` - old ($\sim$ 90's), but easy format to use
- `meshtv/silo` - LLNL software which allows visualization of large datasets associated with large meshes. Can also be used to dump data in checkpoint files which allows for restart computations.

Conclusions

- To obtain the best performance from a computer code, one should know what type of system one is running on and target for optimization areas which can be targeted, i.e., cache locality, SIMD extensions, etc.
- Does the approach work? Complex code adaptive DG FEM for elliptic problem on square domain:
  - No optimization: $\sim$ 37.15 sec.
  - `gcc` compiler opts only: $\sim$ 21.53 sec.
  - `icc` compiler opts only: $\sim$ 19.14 sec.

Conclusions, contd.

- Note that optimizing floating point performance requires the right combination of good programming techniques and compiler optimizations.
- The Intel Pentium4 architechture is not a bad floating point calculational platform, considering it was not designed for that purpose.
- Even better floating point performance can be obtained with CPU's designed for calculations, i.e., 64 bit CPU's, more floating point registers, differently designed caches, etc.
- All of the above tools work very well in the Linux environment and are free!