Serial Code Optimization

Shirley Moore

CPS 5401 Fall 2013

svmoore.pbworks.com

Optimization Process

- Follow best practices when developing code
- Profile code to find “hotspots” (i.e., portions taking the most time)
  - Why?
- Develop performance model for hotspot region
  - Why?
- Do serial optimizations first
  - Why?

Best Practices

- Make code clear and simple
- Access memory in the preferred order (row-major in C, column-major in Fortran)
- Minimize number of complicated math operations (e.g., division, square root)
- Avoid excessive program modularization
  - Use macros
  - Write functions that can be inlined

Best Practices (cont.)

- Avoid type casts and conversions
- Minimize the use of pointers
- Avoid the following within loops
  - Loop-invariant code
  - Branches and conditional statements
  - Function calls and I/O operations

Exploit the Hardware

- Minimize stride
  - Improves cache usage
  - Improves prefetching effectiveness
  - Unit stride is optimal in most cases
  - Power of 2 strides are BAD
- Write loops with independent iterations when possible
  - Can be unrolled and pipelined
  - Can be sent to vector or SIMD units

Array Blocking

- Works with small array blocks when expressions contain mixed stride operations
- Uses complete cache lines
- Avoids eviction that would otherwise happen without blocking
- Example: matrix multiply

Loop Transformations

- Usually best to leave to the compiler but can be done by hand
- Loop unrolling
- Loop fission
- Loop fusion
- Loop interchange

Vectorization

- Write loops with independent iterations so that the compiler can vector
- Uses vector instructions (e.g., SSE, AVX)
- In some cases, an entire loop can be replace with a call to a vector function.

Tools that can help

- Profilers
- Compiler output
- Hardware counters (access using PAPI library or higher-level performance analysis tool)
- Automated performance analysis tools