Introduction to Compilers and Optimization

Le Yan (lyan1@cct.lsu.edu)
Scientific Computing Consultant
Louisiana Optical Network Initiative / LSU HPC
April 1, 2009
Goals of training

• Acquaint users with some of the most frequently used compiler options
• Acquaint users with general optimization concepts and techniques
Outline

• Compiler basics
• Debugging with compilers
• Optimization
  – Overview
  – Compiler optimization
  – Optimized libraries
Outline

• Compiler basics
• Debugging with compilers
• Optimization
  – Overview
  – Compiler optimization
  – Optimized libraries
## Compiler – IBM P5 clusters

<table>
<thead>
<tr>
<th></th>
<th>xl compilers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Serial</strong></td>
<td><strong>MPI</strong></td>
<td></td>
</tr>
<tr>
<td>Fortran</td>
<td>xlFxlf90, xlFxlf95</td>
<td>mpxFxlf, mpxFxlf_r</td>
</tr>
<tr>
<td></td>
<td>xlFxlf_r, xlf90_r_r, xlFxlf95_r</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>xlxlc, xlc_r</td>
<td>mpxCc, mpxCc_r</td>
</tr>
<tr>
<td>C++</td>
<td>xlCxIC_c, xIC_r</td>
<td>mpxCcIC, mpxCc_c</td>
</tr>
</tbody>
</table>

*Note: _r means thread safe*
## Compiler – Dell Linux Clusters

<table>
<thead>
<tr>
<th></th>
<th>Serial</th>
<th>MPI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fortran</td>
<td>C</td>
</tr>
<tr>
<td>Intel</td>
<td>ifort</td>
<td>icc</td>
</tr>
<tr>
<td>PGI</td>
<td>pgf77, pgf90</td>
<td>pgcc</td>
</tr>
</tbody>
</table>

Note: The mpixxx compilers are actually wrappers

```bash
[lyan1@tezpur2 ~]$ mpicc --show
icc -DUSE_STDARG -DHAVE_STDLIB_H=1 -DHAVE_STRING_H=1
-­DHAVE_UNISTD_H=1 -DHAVE_STDARG_H=1 -DUSE_STDARG=1
-­DMALLOC_RET_VOID=1 -L/usr/local/packages/mvapich-1.0-intel10.1/lib -lmptich -L/usr/local/ofed/lib64 -Wl,-rpath=/usr/local/ofed/lib64
-libverbs -libumad -lpthread -lpthread -lrt
```
Usage

- **Serial**
  - `ifort <options> <name_of_source_file>`

- **MPI**
  - `mpicc <options> <name_of_source_file>`

- **OpenMP**
  - **IBM:** `xlc_r <options> -qsmp=omp <name_of_source_file>`
  - **Intel:** `ifort <options> -openmp <name_of_source_file>`
  - **PGI:** `pgf90 <options> -mp <name_of_source_file>`
Basic common options

- **–o**
  - Specifies the name of the output file.
  - **Ex:** `xlc –o test.exe mycode.c`

- **–c**
  - Prevents linking (compile only).
  - **Ex:** `xlc –c –o myobj.o mycode.c`
Basic common options

- `-I`
  - Specifies an additional directory for the include path.
  - **Ex:** `ifort -I/home/lyan1/include -o test.ex mycode.f90`

- `-L`, `-l`
  - Tells the linker to search for specified libraries in a specified directory.
  - **Ex:**
    ```
    ifort -L/usr/local/compilers/Intel/mkl-10.0/lib/em64t
    -lmkl_lapack -lmkl_em64t mycode.f90
    ```
Outline

• Compiler basics
• Debugging with compilers
• Optimization
  – Overview
  – Compiler optimization
  – Optimized libraries
Debugging Tools

- `printf/write` statements
- Compiler (with debugging options enabled)
- Debuggers
  - Symbolic: gdb/idb/dbx/pdbx
  - Graphic: Totalview
Compiler options - debugging

- **–g**
  - Generates full debugging information in the object file.
  - Needed by almost all debuggers

- **–C (xlf, Intel, PGI), –qcheck (xlc)**
  - Enables array bound checking.

- **–traceback, –inline-debug-info (Intel only)**
  - Provides source file trace back information when a severe error occurs at runtime.
Compiler options - debugging

- `-qflttrap` (IBM), `-Ktrap` (pgi), `-fpe0` (Intel)

- Detects floating point exceptions
  - Division by zero
  - Not-a-number values
  - Overflow
  - Underflow
Example

```fortran
[lyan1@tezpur2 debug]$ cat buggy.f90
program buggy
implicit none
real*8 :: a(10),b,c
integer,parameter :: i=-1
integer :: j
! Out-of-bound error at compilation time.
b=a(i)
! Out-of-bound error at run time.
j=i
b=a(j)
! Generate the nanq error.
b=-1.
c=sqrt(b)
stop
end

[lyan1@tezpur2 debug]$ ifort -o buggy buggy.f90
[lyan1@tezpur2 debug]$ ./buggy
[lyan1@tezpur2 debug]$ 
```

- Most compilers do not check for these bugs in the default mode
- This is fair as extra instructions are needed
  - Slow the execution down
With Debugging Option Enabled

```
[lyan1@tezpur2 debug]$ ifort -C -o buggy buggy.f90
fortcom: Error: buggy.f90, line 8: Subscript #1 of the array A has value -1 which is less than the lower bound of 1
b=a(i)
__^
compilation aborted for buggy.f90 (code 1)
```

```
[lyan1@tezpur2 debug]$ vim buggy.f90
[lyan1@tezpur2 debug]$ ifort -C -o buggy buggy.f90
[lyan1@tezpur2 debug]$ ./buggy
forrtl: severe (408): fort: (3): Subscript #1 of the array A has value -1 which is less than the lower bound of 1
```

```
Image          PC              Routine               Line   Source
buggy          0000000000456F56 Unknown          Unknown     Unknown
buggy          0000000000456156 Unknown          Unknown     Unknown
buggy          0000000000418586 Unknown          Unknown     Unknown
buggy          00000000004038A5 Unknown          Unknown     Unknown
buggy          0000000000402B10 Unknown          Unknown     Unknown
buggy          0000000000402948 Unknown          Unknown     Unknown
buggy          00000000004028E2 Unknown          Unknown     Unknown
libc.so.6      00000036B501C3FB Unknown          Unknown     Unknown
buggy          000000000040282A Unknown          Unknown     Unknown
```
Traceback

[lyan1@tezpur2 debug]$ ifort -C -traceback -o buggy buggy.f90
[lyan1@tezpur2 debug]$ ./buggy
fortrl: severe (408): fort: (3): Subscript #1 of the array A has value -1 which is less than the lower bound of 1

<table>
<thead>
<tr>
<th>Image</th>
<th>PC</th>
<th>Routine</th>
<th>Line</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>buggy</td>
<td>0000000000456F56</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>buggy</td>
<td>0000000000456156</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>buggy</td>
<td>0000000000418586</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>buggy</td>
<td>00000000004038A5</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>buggy</td>
<td>0000000000402B10</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>buggy</td>
<td>0000000000402948</td>
<td>MAIN___</td>
<td>11</td>
<td>buggy.f90</td>
</tr>
<tr>
<td>buggy</td>
<td>00000000004028E2</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>libc.so.6</td>
<td>00000036B501C3FB</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>buggy</td>
<td>000000000040282A</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
</tbody>
</table>
Outline

• Compiler basics
• Debugging with compilers
• Optimization
  – Overview
  – Compiler optimization
  – Optimized libraries
Objective of Optimization

- Shorten wall clock time
  - This is how long it takes for your job to run
  - This is how much you get charged
  - This is how long your job will block the jobs of other users

- Use less resources
  - Memory
  - IO
Before you start

• Make sure that the code runs (debugging)
• Find the sections that need to be tuned (profiling)
• Have some simple case to check the correctness against
• Decide what optimization technique to use
  • Hand-tune
  • Compiler optimization options
  • Optimized libraries
Hand-tuning

• Try not to excessively hand-tune your code
  • May make your code hard to read and debug
  • May inhibit compiler optimization

• Hand tuning is necessary when compilers cannot help
  • Example
    • Mathematic consideration
    • Load balancing etc.
Mathematic considerations

- Different operations have different efficiencies
  - Fast: Assignment, Add and Multiplication
  - Slow: Division and Exponential
  - Very slow: Square root
- Try to avoid those slow operations
Mathematic considerations

**Bad**

Do i=1,n
   sum=sum+a(i)/x
enddo

Do i=1,n
   dist=sqrt(a(i))
   if (dist <= cut_off) then
      count=count+1
   endif
enddo

**Good**

inv_x=1./x
Do i=1,n
   sum=sum+a(i)*inv_x
enddo

cut_off_sqr=cut_off*cut_off
Do i=1,n
   if (dist <= cut_off_sqr) then
      count=count+1
   endif
enddo

• In most cases compilers are not smart enough to help you with this
Outline

• Compiler basics
• Debugging with compilers
• Optimization
  – Overview
  – Compiler optimization
  – Optimized libraries
Memory Hierarchy

- The hierarchical structure of storage

<table>
<thead>
<tr>
<th>Access Time (cycle)</th>
<th>Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>$\sim 10^0$</td>
</tr>
<tr>
<td>Cache</td>
<td>$\sim 10^1$</td>
</tr>
<tr>
<td>Memory</td>
<td>$\sim 10^2$</td>
</tr>
<tr>
<td>Disk</td>
<td>$\sim 10^6$</td>
</tr>
</tbody>
</table>

$\sim 10^{12}-10^{15}$
Locality

- Load, execute and store
  - Computation (execution) is fast
  - Data movement (load and store) is slow
  - So our goal is to minimize data movement

- Locality
  - Spatial locality: use data that is close to the location being accessed
  - Temporal locality: re-use data that is being accessed
## Temporal Locality: Scalar Replacement

- Replace array references with scalar variables to improve register usage
- Compiler options
  - `-qhot` (IBM), `-no-scalar-rep` (Intel), `-O3` (PGI)

```plaintext
Do I=1,N
  Do J=1,M
    A(I)=A(I)+B(J)*X(I,J)
  Enddo
Enddo

Do I=1,N
  TMP=A(I)
  Do J=1,N
    TMP=TMP+B(J)*X(I,J)
  Enddo
  A(I)=TMP
Enddo
```
Loop transformation

• Loops are always one of the first targets of optimization

• Types of transformation
  • Blocking
  • Interchange
  • Unroll
  • ...

• Depends on the characteristics of the loops to be transformed
Example: interchange loops

• Minimize stride
  – Left: stride=N
  – Right: stride=1

• Remember: FORTRAN and C are different
  – FORTRAN: column major
  – C: row major
Compiler options

- IBM
  - `–qhot`: controls loop transformations, along with a few other things
- Intel
  - Aggressive loop transformation at `–O3`
- Pgi
  - `–Munroll`
  - `–Mvect`
Optimizing for specific platforms

- Tell the compiler to generate code for optimal execution on a specific platform/architecture
- Main architectural differences
  - Processor design
  - Instruction sets
  - Cache/memory geometry
Optimizing for specific platforms

- **Intel**
  - `-x<processor>`
  - For LONI Linux clusters: `-xT`

- **Pgi**
  - `-tp`: sets the target architecture
  - For LONI Linux clusters: `-tp p7-64`
  - `-Mvect`

- **IBM**
  - `-qarch=pwr5`
  - `-qtune=pwr5`
  - `-qcache=pwr5`
Interprocedural analysis

- Optimize across different files (whole program analysis)
  - Useful when you have a lot of source files

- Compiler options
  - IBM: \(-qipa\)
  - Intel: \(-ip, -ipo\)
  - Pgi: \(-Mipa\)
Inlining

- Reduce call overhead by inlining functions
  - Useful when having many small functions
  - Could result in large executables

- Compiler options
  - **IBM**: `-qipa=inline<=options>`
  - **Intel**: `-inline, -finline-function, -finline-limit`
  - **Pgi**: `-Minline<=options>, -Mextract, -Mrecursive`
Common optimization options

- `-O<n>`
  - Different level of optimization (meaning varies across compilers)
    - IBM: n=0,2,3,4,5
    - Intel: n=0,1,2,3
    - Pgi: n=0,1,2,3,4

- `-fast` (intel and pgi)
  - Intel: `-O3 -xT -ipo -no-prec-div -static`
  - Pgi: `-O2 -Munroll=c:1 -Mnoframe -Mlre`
Outline

• Compiler basics
• Debugging with compilers
• Optimization
  – Overview
  – Compiler optimization
  – Optimized libraries
Optimized Libraries

- **IBM**
  - Mathematical Acceleration Subsystems (MASS)
  - Engineering and Scientific Subroutine Library (ESSL)
- **Intel**
  - Mathematical Kernel Libraries (MKL)
- **PGI**
  - Pre-compiled BLAS and LAPACK
Optimized Libraries

• Chances are that these libraries are much more efficient than code written by ourselves
  • Check these libraries out before writing your own function, especially mathematical operations
  • Some functions from “standard” libraries (e.g. Lapack) might not be implemented in the vendor's library

• There are other scientific libraries other than the ones compiler vendors provide
  • Ex: ATLAS, ARPACK, FFTW, GSL
MASS Library (IBM)

- Usage: link with `-lmass`

- Optimized Intrinsic functions
  - Examples: `sqrt`, `sin`, `cos`, `exp`, `log`
  - Better performance at the expense of reduced precision (1 or 2 bits less)
  - Both scalar and vector versions are available
ESSL (IBM)

- **Usage:** link with `−leSSL`
- **Content**
  - Linear algebra subprograms (BLAS and Lapack)
  - Linear equation solvers
  - Eigen system analysis
  - Fourier transforms
  - Sorting and searching
  - Interpolation
  - Numerical quadrature
  - Random number generator
MKL (Intel)

- Multiple libraries
- Content
  - BLAS and LAPACK
  - ScaLAPACK
  - Fast Fourier transform
  - Vectorized math library
  - Sparse solvers
### Member of MKL libraries

```
[lyan1@tezpur2 ~] $ ls *.a
-r--r--r-- 1 root root  735208 Oct 12 2007 libguide.a
-r--r--r-- 1 root root  763754 Oct 12 2007 libiomp5.a
-r--r--r-- 1 root root  1592158 Oct 12 2007 libmkl_blacs_ilp64.a
-r--r--r-- 1 root root  1593114 Oct 12 2007 libmkl_blacs_intelmpi20_ilp64.a
-r--r--r-- 1 root root  981772 Oct 12 2007 libmkl_blacs_intelmpi20_lp64.a
-r--r--r-- 1 root root  1593114 Oct 12 2007 libmkl_blacs_intelmpi_ilp64.a
-r--r--r-- 1 root root  981772 Oct 12 2007 libmkl_blacs_intelmpi_lp64.a
-r--r--r-- 1 root root  980816 Oct 12 2007 libmkl_blacs_lp64.a
-r--r--r-- 1 root root 1620964 Oct 12 2007 libmkl_blacs_openmpi_ilp64.a
-r--r--r-- 1 root root 1009692 Oct 12 2007 libmkl_blacs_openmpi_lp64.a
-r--r--r-- 1 root root  27 Oct 12 2007 libmkl_cdft.a
-r--r--r-- 1 root root  52214 Oct 12 2007 libmkl_cdft_core.a
-r--r--r-- 1 root root  53172560 Oct 12 2007 libmkl_core.a
-r--r--r-- 1 root root   64 Oct 12 2007 libmkl_em64t.a
-r--r--r-- 1 root root  6067984 Oct 12 2007 libmkl_gf_ilp64.a
-r--r--r-- 1 root root  6350862 Oct 12 2007 libmkl_gf_lp64.a
-r--r--r-- 1 root root  3004602 Oct 12 2007 libmkl_gnu_thread.a
-r--r--r-- 1 root root  6062168 Oct 12 2007 libmkl_intel_ilp64.a
-r--r--r-- 1 root root  6344990 Oct 12 2007 libmkl_intel_lp64.a
-r--r--r-- 1 root root  1712226 Oct 12 2007 libmkl_intel_sp2dp.a
-r--r--r-- 1 root root  4922154 Oct 12 2007 libmkl_intel_thread.a
-r--r--r-- 1 root root   64 Oct 12 2007 libmkl_lapack.a
...
MKL Usage

- Common link option
  - `-L/usr/local/compilers/Intel/mkl-10.0/lib/em64t`

- Libraries to link to
  - BLAS, FFT: `-lmkl_em64t -lmkl_core`
  - LAPACK: `-lmkl_lapack -lmkl_em64t -lmkl_core`
  - ScaLAPACK: `-lmkl_scalpack -lmkl_blacs -lmkl_lapack -lmkl_em64t -lmkl_core`
  - Sparse Solver: `-lmkl_solver -lmkl_lapack -lmkl_em64t -lmkl_core`
Exercise

- Objective: optimize a simple Fortran program
- What the program does
  - Reads coordinate data of a number of points from a file
  - Calculates the distance between each possible pair of points
  - If the distance is smaller than 1, then calculate an energy as a function of the distance and add it to the total energy
  - Print the total energy and the number of contributing pairs to screen as the final output
Exercise

• What you need to do
  • Copy the program and data file to your user space
    • `cp /home/lyan1/traininglab/opt/paircount.* <path_to_your_dir>`
  • Get the shortest wall time
    • Use different compiler options
    • Hand-tune the program
Questions?