Cray X1 Tuning for Performance Using Compilation Options

Terry Greyzck, Cray Inc.  CUG 2005
Introduction

- Fortran, C, and C++ compilers
- Common optimizer and code generator
- Fortran and C can have different command line options for historical reasons
- Careful use of compilation options can improve application performance significantly
Default Optimizations

- **Automatic parallelization at multiple levels:**
  - Automatic SIMD parallelism (vectorization)
  - Automatic MIMD parallelism (multistreaming)
  - Scheduling, register allocation, decoupled vector and scalar

- **Aggressive performance enhancements:**
  - Automatic limited interprocedural analysis
  - Aggressive floating point optimizations
  - Somewhat aggressive scalar optimization
Tuning Through the Command Line

• Default optimization is roughly equivalent to:
  -Oscalar2 –Ovector2 –Ostream2 –Oipa3 –Ofp2

• C and C++ use different options:
  -hscalar2 –hvector2 –hstream2 –hipa3 –hfp2

• These options, and others, can be modified to improve performance for most applications
Increasing Optimization:  \(-{Oscalar3}\)

- C and C++ option:  \(-{hscalar3}\)
- Scalar optimization refers to transformations not directly related to parallelization
- Can affect the speed of parallel code (Amdahl’s Law)
- New optimizations are performed at this level, and more aggressive versions of the default ones
- Safe, but possible differences due to reassociation
- From no gain, to 15% or more
Increasing Optimization:  \texttt{-Ovector3}

- C and C++ option:  \texttt{-hvector3}
- More extensive dependency analysis
- More aggressive array privatization analysis
- Speculative loading of unsafe memory references
- Safe to use
Increasing Optimization: -O3

- Using –O3 is shorthand for:
  - Fortran:
    -Oscalar3 –Ovector3 –Ostream3
  - C and C++:
    -hscalar3 –hvector3 –hstream3
  - Other options need to be specified individually
Increasing Optimization: -Ofp3

- C and C++ option: -hfp3
- Higher levels of $fp$ give the compiler more freedom to optimize floating point expressions
- Floating Point (fp) optimization control
- Inline code for natural log, exponentiation, and power functions
- Use if the application has some tolerance for floating point differences
-Oaggress

- C and C++ option: -haggress
- Tells the compiler to avoid regioning procedures and functions
- Most functions are not regioned in the first place
- Disabling regioning for large functions can dramatically increase the compilation time
- Performance gain is minimal; this option is generally not beneficial
Decreasing Performance: \textit{-O}overindex

- C and C++ option: \textit{-hoverindex}
- Useful for hand-collapsed loops and similar codes of questionable legality
- Serious negative performance impact
- For the \textit{Perfect} suite, 1,243 of 4,613 vectorized loops (27\%) are hurt by this option
- Shuts down much of scheduling and memory ordering optimizations
- It is better to correct the code to comply with the language standard
Decreasing Performance: \(-Ofp0, -Ofp1\)

- C and C++ options: \(-hfp0, -hfp1\)
- Lower levels restrict the optimization of floating point expressions
- Correspondingly lower performance can result
- These options should be avoided if possible
Decreasing Performance: \(-Oipa5\)

- C and C++ option: \(-hipa5\)
- Inlines \textit{everything, everywhere}
- Can dramatically increase compilation time and executable size
- Can actually \textit{decrease} overall performance
- This option should never be used
- The default \((-Oipa3)\) is best for most codes
Other Options to Avoid

<table>
<thead>
<tr>
<th>Fortran</th>
<th>C and C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>-eh</td>
<td></td>
</tr>
<tr>
<td>-ev</td>
<td></td>
</tr>
<tr>
<td>-e0</td>
<td>-hzero</td>
</tr>
<tr>
<td>-Onoinfinitevl</td>
<td>-hnoinfinitevl</td>
</tr>
<tr>
<td>-Onorecurrence</td>
<td>-hnorecurrence</td>
</tr>
<tr>
<td>-Ounroll0</td>
<td>-hnounroll</td>
</tr>
<tr>
<td>-Oshortcircuit2</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fortran</th>
<th>C and C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>-hzero</td>
<td>-hnointerchange</td>
</tr>
<tr>
<td>-hnofusion[0,1]</td>
<td>-hnofusion</td>
</tr>
<tr>
<td>-hnointerchange</td>
<td>-hnointerchange</td>
</tr>
<tr>
<td>-hnovsearch</td>
<td>-hnovsearch</td>
</tr>
<tr>
<td>-Ozeroinc</td>
<td>-hzeroinc</td>
</tr>
<tr>
<td>-eL, -ew</td>
<td></td>
</tr>
</tbody>
</table>
Voiding Your Warranty:  \texttt{-hivdep}

- Only available for C and C++
- Places an implicit \texttt{ivdep} directive (ignore vector dependencies) on every loop in the code
- Holdover from an earlier compiler
- Can lead to incorrect \textit{and} slower results!
- Never use it. Ever.
Providing Information:  

- C and C++ option:  \(-hssp\)
- Disables multistreaming
- Use on a case by case basis; if there is little MIMD (multistream) parallelism, then running in SSP mode may make better overall use of machine resources
- Gnu utilities are a good example of applications that do not parallelize well
Choosing Between MSP and SSP

• How do you decide when to use \(-Ossp\)?
• One method: Run the application twice, once built with default options, and once built with \(-Ossp\). Time both of them, and pick the fastest
• If the times are close, keep in mind that running in SSP mode frees up the other three processors of the MSP for other applications to use
Providing Information:  `-hrestrict=f`

- This option is available to C and C++
- Gives pointers which are function parameters roughly the same aliasing attributes that Fortran dummy arguments have
- Misuse can result in incorrect programs
- Many codes are written in restrained and compiler-friendly manners, and can use this option
Recommended Command Line

- The simplest command line options with the greatest impact:
  - Fortran: `-O3 -Ofp3`
  - C and C++: `-O3 -hfp3`
- Cray command lines are kept as short and simple as possible
- Avoid options that decrease performance
- Floating point sensitive codes may have to remove the `fp3` option
- If increased compilation time is a problem, the `-O3` option may also need to be removed