Profiling Applications for Sequential and Parallel Performance

Carlos P Sosa
IBM and Biomedical Informatics & Computational Biology,
University of Minnesota Rochester
Minneapolis, MN
Agenda

- Motivation
- What is performance tuning?
- Speedup and efficiency
- Silver an IBM JS22
- Compiling with the XL compilers
  - time
- Identifying Performance Bottlenecks
  - Xprofiler (Xprof)
  - hpmcout
  - MPI Trace
What is the Problem?
Biomedical Informatics & Computational Biology

http://www.reed-electronics.com/electronicnews/article/CA508575.html?industryid=21365
From Kilobytes to Petabytes in 50 Years: http://www.eurekalert.org/features/doe/2002-03/dlnl-fkt062102.php
“The Looming Petascale”

“Chemists gear up for a new generation of supercomputers”

“The new petascale computers will be 1,000 times faster than the terascale supercomputers of today, performing more than 1,000 trillion operations per second. And instead of machines with thousands of processors, petascale machines will have many hundreds of thousands that simultaneously process streams of information.”

“This technological sprint could be a great boon for chemists, allowing them to computationally explore the structure and behavior of bigger and more complex molecules.”
What is the Catch?
Biomedical Informatics & Computational Biology

Applications ...

... are we there?
• What is performance tuning?
• Speedup and efficiency
• Silver an IBM JS22
• Compiling with the XL compilers
  – time
• Identifying Performance Bottlenecks
  – Xprofiler (Xprof)
  – hpmcout
  – MPI Trace
What is Performance Tuning?

Application (software) optimization is the process of making it work more efficiently:
- Executes faster
- Uses less memory
- Performs less I/O
- Better use of resources

Robert Sedgewick, *Algorithms*, 1984, p. 84

Application Flow Analysis

Biomedical Informatics & Computational Biology

Tasks

Work

Time

12/6/2010
Application Optimization

Application performance analysis

- Memory bound?
- I/O bound?
- CPU bound?
1. Tune for compiler optimization flags
2. Locate hot-spots in the code
3. Use highly tuned libraries MASS/ESSL
4. Manually optimize the code
5. Determine if I/O plays a role and tune if needed
• What is performance tuning?
• **Speedup and efficiency**
• Silver an IBM JS22
• Compiling with the XL compilers
  – time
• **Identifying Performance Bottlenecks**
  – Xprofiler (Xprof)
  – hpmcout
  – MPI Trace
Two Key Concepts

• Speedup

• Efficiency
• Speedup is defined as the ratio between the run time of the original code and the run time of the modified code.

\[
\text{Speedup} = \frac{\text{Original code run time}}{\text{Modified code run time}}
\]
Parallel speedup is defined as the ratio between the run time of the sequential code and the run time of the modified code.

\[
\text{Parallel Speedup} = \frac{\text{Sequential run time}}{\text{Parallel run time}}
\]

Run time is measured as elapsed time (or wallclock).
• Parallel efficiency is defined as how well a program (your code) utilizes multiple processors (cores)

\[
\text{Efficiency} = \frac{\text{Sequential run time}}{N_{\text{processors}} \times \text{Parallel run time}}
\]

N is the number of processors defined by the user
Parallel Efficiency Dependencies

Sequential code
Parallel code
Communication (overhead and redundancy)
Example: Parallel Speedup

*Completion time = computation time + communication time*

<table>
<thead>
<tr>
<th>Processors</th>
<th>Serial</th>
<th>Parallel</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>Programmer A</td>
<td>4</td>
<td>25</td>
<td>4</td>
</tr>
<tr>
<td>Programmer B</td>
<td>4</td>
<td>35</td>
<td>2.9</td>
</tr>
<tr>
<td>Programmer c</td>
<td>4</td>
<td>45</td>
<td>2.2</td>
</tr>
</tbody>
</table>
Optimization Comparison

Time reduction

Processors

Time

1 4

Programmer A
Programmer B
Programmer C

2.9x
2.2x
4x
• What is performance tuning?
• Speedup and efficiency
• **Silver an IBM JS22**
• Compiling with the XL compilers
  - time
• **Identifying Performance Bottlenecks**
  - Xprofiler (Xprof)
  - hpmcout
  - MPI Trace
Silver = IBM JS22

Biomedical Informatics & Computational Biology

https://www.msi.umn.edu/labs/umbcl/

- Six JS22 blades each with four 4.0 GHz Power6 processors and 8 GB of memory
- One QS22 blade with two 3.2 GHz PowerXCell 8i processors and 16 GB of memory
<table>
<thead>
<tr>
<th><strong>Blades</strong></th>
<th><strong>JS22</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor Technology</td>
<td>POWER6</td>
</tr>
<tr>
<td># of Cores</td>
<td>4</td>
</tr>
<tr>
<td>GHz</td>
<td>4.0 GHz</td>
</tr>
<tr>
<td>Memory (Min/Max)</td>
<td>2/32 GB</td>
</tr>
<tr>
<td>Local Disk (Min/Max)</td>
<td>0/1</td>
</tr>
<tr>
<td></td>
<td>73/146 GB</td>
</tr>
<tr>
<td>Base Ethernet Ports (1 GB)</td>
<td>2</td>
</tr>
<tr>
<td>Expansion Cards</td>
<td>0-2</td>
</tr>
<tr>
<td>Virtualization</td>
<td>VIOS/IVM</td>
</tr>
<tr>
<td># of Partitions (max)</td>
<td>40</td>
</tr>
<tr>
<td>AIX</td>
<td>Yes</td>
</tr>
<tr>
<td>Linux</td>
<td>Yes</td>
</tr>
<tr>
<td>i Release</td>
<td>6.1</td>
</tr>
<tr>
<td>Processor Tier</td>
<td>P10</td>
</tr>
<tr>
<td>Rperf</td>
<td>30.3</td>
</tr>
<tr>
<td>CPW</td>
<td>13,800</td>
</tr>
<tr>
<td>Chassis*</td>
<td>BladeCenter S, H, HT</td>
</tr>
</tbody>
</table>
One 2.5” SFF SAS HDD up to 146GB total storage on Blade

Redundant connections to BladeCenter chassis

Two 64-bit POWER6 Processors dual core @ 4.0GHZ

Four DIMM slots for up to 32GB DDR2 667 MHz memory

Optional I/O connector
Memory Hierarchy

Registers → L2 cache → DRAM

Increase in memory latency
• What is performance tuning?
• Speedup and efficiency
• Silver an IBM JS22
• Compiling with the XL compilers
  – Time
    • Lab 1
• Identifying Performance Bottlenecks
  – Xprofiler (Xprof)
  – hpmcout
  – MPI Trace
IBM XL compilers are the result of a joint effort:
- IBM Toronto Lab
- Yorktown Heights Research Lab
- Other IBM facilities

Collaboration began in the 1980s with the first POWER chips

XL Fortran for AIX compiler in 1989, and subsequently, the C and C++ for AIX compilers in 1991

**COMPILER:**
The optimizer includes five base optimization levels; 
- `-O0`, `-O2`, `-O3`, `-O4`, and `-O5`

These levels allow you to choose from minimal optimization to intense program analysis.

Optimization analyses range from local basic block to subprogram to file-level to whole-program analysis.

The higher the optimization level, the more intense the program analysis becomes as increasingly sophisticated optimization techniques are applied to your code.

At any optimization level, the compiler performs transformations that result in performance improvements, while still executing your code the way it was written.

At higher levels, the compiler can trade numeric precision for execution speed.

If this effect is not desired, you can specify compiler options such as `-qstrict` to prevent such trade-offs.
What is Optimization?

- Optimization is a compiler driven process that searches for opportunities to restructure your source code and give your application better overall performance.

- The XL compiler optimization performs best on well-written source code that has already been through a thorough debugging and testing process.

- Optimization transformations can:
  - Reduce the number of instructions your application executes to perform critical operations.
  - Restructure your object code to make optimal use of the PowerPC architecture.
  - Improve memory subsystem usage.
  - Exploit the ability of the architecture to handle large amounts of shared memory parallelization.
• Not all optimizations benefit all applications
  - Basic optimization techniques can result in a performance benefit
What is Tuning?

• Tuning offers opportunities to adjust characteristics of your application to improve performance, or to target specific execution environments.

• Even at low optimization levels, tuning for your application and target architecture can have a positive impact on performance.

• With proper tuning the compiler can:
  - Select more efficient machine instructions.
  - Generate instruction sequences that are more relevant to your application.
Initial Steps in Optimization

1. The Basic optimization step begins your optimization processes at levels 0 and 2

2. The Advanced optimization step exposes your application to more intense optimizations at levels 3 through 5

3. The High-order transformation (HOT) step can help you limit loop execution time
• Ensure that your application compiles and executes properly at low optimization levels before trying more aggressive optimizations

<table>
<thead>
<tr>
<th>Optimization level</th>
<th>Additional options implied by default</th>
<th>Complementary options</th>
<th>Other options with possible benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O0</td>
<td>None</td>
<td>-qarch</td>
<td>-g</td>
</tr>
<tr>
<td>-O2</td>
<td>-qmaxmem=8192</td>
<td>-qarch -qtune</td>
<td>-qmaxmem=-1 -qhot=level=0</td>
</tr>
</tbody>
</table>
Level 0 Optimization

Biomedical Informatics & Computational Biology

• **Benefits at level 0**
  - Minimal performance improvement, with minimal impact on machine resources
  - Exposes some source code problems, helping in the debugging process

• Begin your optimization process at `-O0`

• Additionally, specifying `-qarch` at this level targets your application for a particular machine and can significantly improve performance by ensuring your application takes advantage of all applicable architectural benefits
• **Benefits at level 2**
  - Eliminates redundant code
  - Basic loop optimization
  - Can structure code to take advantage of `-garch` and `-qtune` settings
• Higher optimization levels can have a tremendous impact on performance, but

• some trade-offs:
  – Code size
  – Compile time
  – Resource requirements
  – Numeric or algorithmic precision
<table>
<thead>
<tr>
<th>Optimization Level</th>
<th>Additional options implied</th>
<th>Complementary options</th>
<th>Options with possible benefits</th>
</tr>
</thead>
<tbody>
<tr>
<td>-O3</td>
<td>-qnostrict</td>
<td>-qarch</td>
<td>-qpdf</td>
</tr>
<tr>
<td></td>
<td>-qmaxmem=-1</td>
<td>-qtune</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-qhot=level=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-O4</td>
<td>-qnostrict</td>
<td>-qarch</td>
<td>-qpdf</td>
</tr>
<tr>
<td></td>
<td>-qmaxmem=-1</td>
<td>-qtune</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-qhot</td>
<td>-qcachfe</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-qipa</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-qarch=auto</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-qtune=auto</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>-qcachfe=auto</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Level 3 Optimization

- In-depth “Aliasing”
- Better loop scheduling
- High-order loop analysis and transformations (-qhot=level=0)
- Inlining of small procedures within a compilation unit by default
- Eliminating implicit compile-time memory usage limits
- Widening, which merges adjacent load/stores and other operations
- Pointer aliasing improvements to enhance other optimizations
How to login from a Windows Laptop

Step 1

Step 2

http://static.msi.umn.edu/user_support/xclient/xwin_config.html
Putty Session lab 1-2

login as: cpsosa
Using keyboard-interactive authentication.
Password:
Last login: Thu Dec 3 22:30:02 2009 from 24.131.169.94

University of Minnesota Supercomputing Institute
MSI-UMR BICB Computational Laboratory

For assistance please contact us at http://www.msi.umn.edu/consult.html, help@msi.umn.edu, or (612)626-0802.

The available scratch space on Silver is /scratch1. All files in the scratch directory that have not been modified for 14 days will be deleted.

cpsosa@silver:~>
Files Location on silver: lab 1-3

Biomedical Informatics & Computational Biology

silver>cd /scratch1
silver>mkdir temp10
silver>cd temp10

silver>cp -r /scratch1/cpsosa/bicb8510 .

subdirectories:

```
/c
/dgamm /hello /mpi
hello.c
helloWorld.c
pi.c
/cpp
/dgamm /hello /mpi
hello.cpp
/matmul.f
/hello.f
/fortran
/dgamm /hello /mpi
matmul.f
```
• IBM's native compilers (xlf and xlc) and GNU compilers are available on silver
  – cpsosa@silver:~> module avail
  – cpsosa@silver:~> module load xlf
  – cpsosa@silver:~> module load xlc

• Compiling:
  – cpsosa@silver:~> xlf -O3 -qtune=pwr6 -qarch=pwr6 prog.f
  – cpsosa@silver:~> xlc -O3 -qtune=pwr6 -qarch=pwr6 prog.c
  – cpsosa@silver:~> xlC -O3 -qtune=pwr6 -qarch=pwr6 prog.cpp
Using UNIX "time" Command: lab 1-5

Hello World!
Welcome to HPC

real 0.00
user 0.00
sys 0.00

Timing

More info: http://www.ncsu.edu/itd/hpc/Documents/sprofile.php#system

12/6/2010
• What is performance tuning?
• Speedup and efficiency
• Silver an IBM JS22
• Compiling with the XL compilers
• Identifying Performance Bottlenecks: High Performance Toolkit
  - Xprofiler
    • Lab 2
High Performance Toolkit

- High Performance Computing Toolkit
  - Xprofiler for CPU profiling
  - Hardware Performance Monitoring (HPM)
  - Message Passing Interface (MPI) Profiler and Tracer tool
  - I/O Performance
- GUI of the High Performance Computing Toolkit (HPCT)
High Level Design Flow for HPCS Toolkit

- HPCS Toolkit provides Framework for Performance Analysis
  - Performance evaluation and decision system
  - Capability with graphical/visual interface always available

Bottleneck: elapsed time exceeds threshold for completing work
CPU Profiling using Xprofiler

• **Xprofiler:**
  - Used to analyze your application performance
  - It uses data collected by the `-pg` compiler option to construct a graphical display
  - It identifies functions that are the most CPU intensive

• **GUI** manipulates the display in order to focus on the critical areas of the application
Starting Xprofiler

Biomedical Informatics & Computational Biology

• Start Xprofiler by issuing the Xprofiler command from the command line
  - Specify the executable
  - Profile data file or files
  - Options
    • Specify them on the command line, with the Xprofiler command
    • Issue the Xprofiler command alone and then specify the options from within the GUI

• $Xprofiler a.out gmon.out... [options]
  - a.out is the name of your binary executable file
  - gmon.out is the name of your profile data file or files
  - options
• Xprofiler gives a graphical picture of the CPU consumption of your application in addition to textual data
• Xprofiler displays your profiled program in a single main window
• It uses several types of graphic images to represent the relevant parts of your program:
  - Functions are displayed as solid green boxes, called function boxes
  - Calls between them are displayed as blue arrows, called call arcs
  - The function boxes and call arcs that belong to each library within your application are displayed within a fenced-in area called a cluster box
• When Xprofiler first opens, by default, the function boxes for your application are clustered by library. This type of clustering means that a cluster box appears around each library, and the function boxes and call arcs within the cluster box are reduced in size.
  - If you want to see more detail, you must uncluster the functions by selecting File → Uncluster Functions
**Xprofiler Main Menus**

- **File menu**
  - With the File menu, you specify the executable (a.out) files and profile data (gmon.out) files that Xprofiler will use. You also use this menu to control how your files are accessed and saved.

- **View menu**
  - You use the View menu to help you focus on portions of the function call tree, in the Xprofiler main window, in order to have a better view of the application's critical areas.

- **Filter menu**
  - Using the Filter menu, you can add, remove, and change specific parts of the function call tree. By controlling what Xprofiler displays, you can focus on the objects that are most important to you.

- **Report menu**
  - The Report menu provides several types of profiled data in a textual and tabular format. With the options of the Report menu, you can display textual data, save it to a file, view the corresponding source code, or locate the corresponding function box or call arc in the function call tree, in addition to presenting the profiled data.

- **Utility menu**
  - The Utility menu contains one option, *Locate Function By Name*, with which you can highlight a particular function box in the function call tree.
• **Function menu**
  - Number of operations for any of the functions shown in the function call tree by using the Function menu. You can access statistical data, look at source code, and control which functions are displayed.
  - The Function menu is not visible from the Xprofiler window. To access it, you right-click the function box of the function in which you are interested.

• **Arc menu**
  - Locate the caller and callee functions for a particular call arc.
  - The Arc menu is not visible from the Xprofiler window. You access it by right-clicking the call arc in which you are interested.

• **Cluster Node menu**
  - Control the way your libraries are displayed by Xprofiler.
  - The Cluster Node menu is not visible from the Xprofiler window. You access it by right-clicking the edge of the cluster box in which you are interested.
  - Display Status Field at the bottom of the Xprofiler window is a single field that tells you:
    • The name of your application
    • The number of gmon.out files used in this session
    • The total amount of CPU used by the application
    • The number of functions and calls in your application and how many are currently displayed
# LOADER/LINKER:
# Use Standard options
setenv LOAD "xlf90 -pg -bmaxdata:0x80000000 "
# Load with the IBM MASS & ESSL libraries
setenv LOADLIB ""
if ( $HAS_MASSLIB == "yes" ) setenv LOADLIB "-L$MASSLIBDIR -lmassvp4 "
if ( $VENDOR_BLAS == "yes" ) setenv LOADLIB "$LOADLIB -lblas "
if ( $VENDOR_LAPACK == "yes" ) setenv LOADLIB "$LOADLIB -lessl "

# little or no optimization:
setenv L0 "xlf90 -pg -qfixed -c"
# modest optimization (local scalar):
setenv L1 "xlf90 -pg -qfixed -O2 -c"
# high scalar optimization (but not vectorization):
setenv L2 "xlf90 -pg -qfixed -O3 -qmaxmem=-1 -qarch=auto -qtune=auto -c"
# high optimization (may be vectorization, not parallelization):
setenv L3 "xlf90 -pg -qfixed -O3 -qmaxmem=-1 -qarch=auto -qtune=auto -c"
Xprofiler Calling Tree

Function boxes

Call arcs
Program: sander  Total CPU Usage: 30.03 seconds (summary of 1 gmon.out profile files)
Display Status: showing 210 out of 210 nodes and 217 out of 217 arcs
Functions are represented by green, solid-filled boxes in the function call tree:
- The *size and shape* of each function box indicates its CPU usage.
- The *height* of each function box represents the amount of CPU time it spent on executing itself.
- The *width* of each function box represents the amount of CPU time it spent on executing itself, plus its descendant functions.

**Function, cycle, total amount of CPU time** (in seconds) this function spent on itself plus descendants (the number to the left of the x), the amount of CPU time (in seconds) this function spent only on itself (the number to the right of the x).

Call *arc labels* show the number of calls that were made between the two functions (from caller to callee).
Library Filters (before)
Library Filters (after)
Looking at the Source Code

Biomedical Informatics & Computational Biology

<table>
<thead>
<tr>
<th>File</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>no, ticks</td>
<td>source code</td>
</tr>
<tr>
<td>line</td>
<td>per line</td>
</tr>
<tr>
<td>230</td>
<td>DO 50, I = 1, M</td>
</tr>
<tr>
<td>231</td>
<td>C ( I, J ) = ZERO</td>
</tr>
<tr>
<td>232</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>233</td>
<td>ELSE IF ( BETA, NE, ONE ) THEN</td>
</tr>
<tr>
<td>234</td>
<td>DO 80, I = 1, M</td>
</tr>
<tr>
<td>235</td>
<td>1</td>
</tr>
<tr>
<td>236</td>
<td>C ( I, J ) = BETA * C ( I, J )</td>
</tr>
<tr>
<td>237</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>238</td>
<td>END IF</td>
</tr>
<tr>
<td>239</td>
<td>DO 80, L = 1, K</td>
</tr>
<tr>
<td>240</td>
<td>IF ( B ( L, J ), NE, ZERO ) THEN</td>
</tr>
<tr>
<td>241</td>
<td>TEMP = ALPHA * B ( L, J )</td>
</tr>
<tr>
<td>242</td>
<td>1</td>
</tr>
<tr>
<td>243</td>
<td>850</td>
</tr>
<tr>
<td>244</td>
<td>70</td>
</tr>
<tr>
<td>245</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>246</td>
<td>END IF</td>
</tr>
<tr>
<td>247</td>
<td>ELSE</td>
</tr>
<tr>
<td>248</td>
<td>80</td>
</tr>
<tr>
<td>249</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>250</td>
<td>ELSE</td>
</tr>
<tr>
<td>251</td>
<td>90</td>
</tr>
<tr>
<td>252</td>
<td>CONTINUE</td>
</tr>
</tbody>
</table>

Tick marks

Search Engine: (regular expressions supported)

ijdgemm

./matmul
Looking at Assembler Code

Biomedical Informatics & Computational Biology

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Assembler Code</th>
<th>Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>2290</td>
<td>7080298</td>
<td>xfer 0, 8</td>
<td></td>
</tr>
<tr>
<td>2294</td>
<td>7080028</td>
<td>xfer 12</td>
<td></td>
</tr>
<tr>
<td>2298</td>
<td>48016F0</td>
<td>beq</td>
<td></td>
</tr>
<tr>
<td>229C</td>
<td>F0160001</td>
<td>std 31, 0x3fe(1)</td>
<td></td>
</tr>
<tr>
<td>22A0</td>
<td>F0160001</td>
<td>std 30, 0x3fe(1)</td>
<td></td>
</tr>
<tr>
<td>22A4</td>
<td>81F0000</td>
<td>or1</td>
<td>31, 10, 0</td>
</tr>
<tr>
<td>22A8</td>
<td>F0160001</td>
<td>std 29, 0x3fe(1)</td>
<td></td>
</tr>
<tr>
<td>22AC</td>
<td>F0160001</td>
<td>std 28, 0x3fe(1)</td>
<td></td>
</tr>
<tr>
<td>22B0</td>
<td>81E0000</td>
<td>or1</td>
<td>28, 0, 0</td>
</tr>
</tbody>
</table>

Program: natmul  Total CPU Usage: 39.33 seconds (summary of 1 slow, out profile files)
Display Status: showing 6 out of 75 nodes and 5 out of 70 areas
<table>
<thead>
<tr>
<th>%time</th>
<th>cumulative seconds</th>
<th>self seconds</th>
<th>calls</th>
<th>ms/call</th>
<th>ms/call</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>55.0</td>
<td>16.53</td>
<td>16.53</td>
<td>235580</td>
<td>0.07</td>
<td>0.07</td>
<td>.short_ene [7]</td>
</tr>
<tr>
<td>9.1</td>
<td>19.27</td>
<td>2.74</td>
<td>23558</td>
<td>0.12</td>
<td>0.12</td>
<td>.pack_nb_list [11]</td>
</tr>
<tr>
<td>8.1</td>
<td>21.71</td>
<td>2.44</td>
<td>10</td>
<td>244.00</td>
<td>244.00</td>
<td>.grad_sumrc [12]</td>
</tr>
<tr>
<td>6.2</td>
<td>23.57</td>
<td>1.86</td>
<td>10</td>
<td>186.00</td>
<td>190.00</td>
<td>.fill_charge_grid</td>
</tr>
</tbody>
</table>
Mathematical Acceleration Subsystem (MASS) consists of libraries of tuned mathematical intrinsic functions.

Scalar Library: The MASS scalar library, libmass.a, contains an accelerated set of frequently used math intrinsic functions in the AIX and Linux system library libm.a (now called libxlf90.a in the IBM XL Fortran manual): sqrt, rsqrt, exp, log, sin, cos, tan, atan, atan2, sinh, cosh, tanh, dnint, x**y

Vector Library: The general vector library, libmassv.a, contains vector functions that will run on the entire IBM pSeries and Blue Gene families.
c------------------------------------------------------------------------
c     Loop over the 12-6 LJ terms for eedmeth = 1
c------------------------------------------------------------------------

c icount = 0
do m = 1,numvdw
   include "ew_directp.h"
enddo
c
c calculation starts: loop over the data gathered in the temporary
array.
c
C*$* NO FUSION
do im_new = 1,icount

   j = tempint(im_new)
delr2 = tempre(5*im_new)
c
c -- cubic spline on switch:
```fortran

c delrinv = 1.0/sqrt(delr2)
delr = delr2*delrinv
delr2inv = delrinv*delrinv
x = dxdr*delr
ind = edtbdns*x
dx = x - ind*del
ind = 4*ind

e3dx = dx*eed_cub(3+ind)
e4dx = dx*dx*eed_cub(4+ind)
switch = eed_cub(1+ind) + dx*(eed_cub(2+ind) +
$(e3dx + e4dx*third)*half

d_switch_dx = eed_cub(2+ind) + e3dx + e4dx*half
```
c--------------------------------------------------------------------------
c     Loop over the 12-6 LJ terms for eedmeth = 1
 c--------------------------------------------------------------------------
c
    icount = 0
do m = 1,numvdw
     include "ew_directp.h"
 enddo

c calculation starts: loop over the data gathered in the temporary
array caches.
c
#define MASSLIB
  call vrsqrt( cache_df, cache_r2, icount )
#else
    do im_new = 1,icount
       delr2 = cache_r2(im_new)
delrinv = 1.0/sqrt(delr2)
cache_df(im_new) = delrinv
 enddo
#endif
do im_new = 1, icount

    j = cache_bckptr(im_new)
delr2 = cache_r2(im_new)
delrinv =
    cache_df(im_new)
c
    -- cubic spline on
switch:
c
    delr = delr2*delrinv
delr2inv =
    delrinv*delrinv
    x = dxdr*delr
    ind = eedtbdns*x
dx = x - ind*del
    ind = 4*ind
### Single processor Optimization

**Biomedical Informatics & Computational Biology**

#### without MASS  with MASS

<table>
<thead>
<tr>
<th></th>
<th>without MASS</th>
<th>with MASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed</td>
<td>2579.95</td>
<td>2226.20</td>
</tr>
<tr>
<td>User</td>
<td>2574.65</td>
<td>2224.06</td>
</tr>
<tr>
<td>Sys</td>
<td>0.50</td>
<td>0.47</td>
</tr>
</tbody>
</table>

- POWER 375 MHz
- 15% Speedup

vector mass
%cd/scratch1/cpsosa/bicb8510/fortran/dgemm

%module load xlf

%make -f make.ibm1

xlf -pg -c -O3 -qhot -qarch=pwr6 -qtune=pwr6 -q64 matmul.f
** matmul  === End of Compilation 1 ===
1501-510 Compilation successful for file matmul.f.

xlf -pg -c -O3 -qhot -qarch=pwr6 -qtune=pwr6 -q64 dgemm.f
** dgemm  === End of Compilation 1 ===
"dgemm.f", 1500-036 (I) The NOSTRICT option (default at OPT(3)) has the potential to alter the semantics of a program. Please refer to documentation on the STRICT/NOSTRICT option for more information.

1501-510 Compilation successful for file dgemm.f.

xlf -pg -c -O3 -qhot -qarch=pwr6 -qtune=pwr6 -q64 lsame.f
** lsame  === End of Compilation 1 ===
1501-510 Compilation successful for file lsame.f.

xlf -pg -c -O3 -qhot -qarch=pwr6 -qtune=pwr6 -q64 xerbla.f
** xerbla  === End of Compilation 1 ===
1501-510 Compilation successful for file xerbla.f.

xlf -pg -o matmul -q64 matmul.o dgemm.o lsame.o xerbla.o
Running Xprofiler on Silver: lab 2-2

%.matmul
mflops are 964.993481618238206

%ls
dgemm.f gmon.out lsame.o matmul
matmul.o xerbla.o
dgemm.o lsame.f make.ibm1 matmul.f
xerbla.f

%module load hpct
%Xprof ./matmul gmon.out
Running Xprofiler on Silver: lab 2-3

Biomedical Informatics & Computational Biology
• What is performance tuning?
• Speedup and efficiency
• Silver an IBM JS22
• Compiling with the XL compilers

**Identifying Performance Bottlenecks:**

- `hpmcount`
  - Lab 3
• **Hardware Performance Counter:**
  - Set of special-purpose "registers" built into modern microprocessors to store the counts of hardware-related activities within computer systems.
  - Advanced users often rely on those counters to conduct "low-level performance analysis or tuning."

http://en.wikipedia.org/wiki/Hardware_performance_counter

"To understand what happens inside a processor when an application is executed, processor architects designed a set of special registers to count the events taking place when processors are executing instructions."

12/6/2010
• **Processor register (or general purpose register)** is a small amount of storage available on the CPU whose contents can be accessed more quickly than storage available elsewhere. [http://en.wikipedia.org/wiki/Processor_register](http://en.wikipedia.org/wiki/Processor_register)

• **Microprocessor** incorporates most or all of the functions of a central processing unit (CPU) on a single integrated circuit (IC). [http://en.wikipedia.org/wiki/Microprocessor](http://en.wikipedia.org/wiki/Microprocessor)

• **Performance tuning** is the improvement of system performance. [http://en.wikipedia.org/wiki/Performance_tuning](http://en.wikipedia.org/wiki/Performance_tuning)
Software Profilers versus Hardware Counters

- Hardware counters provide low-overhead access to a wealth of detailed performance information related to CPU's functional units, caches and main memory.
- With hardware counters no source code modifications are needed in general.
- Meaning of hardware counters vary from one kind of architecture to another due to the variation in hardware organizations.
- Difficulties correlating the low level performance metrics back to source code.
- Limited number of registers to store the counters often force users to conduct multiple measurements to collect all desired performance metrics.
- Modern superscalar processors schedule and execute multiple instructions at one time.

http://en.wikipedia.org/wiki/Hardware_performance_counter
• hpmcount command provides:
  – Execution wall clock time
  – Hardware performance counters information
  – Derived hardware metrics
  – Resource utilization statistics (obtained from the getrusage() system call) for the application named by command
hpmcount [options]

- a  Aggregates the counters on POE runs
- d  Adds detailed set counts for counter multiplexing mode
- H  Adds hypervisor activity on behalf of the process
- h  Displays help message
- k  Adds system activity on behalf of the process
- o  file Output file name
- s  set Lists a predefined set of events or a comma-separated list of sets (1 to N, or 0 to select all.

• To run the ls command and write information concerning events in set 5 from hardware counters, enter:
  - hpmcount -s 5 ls

• To run the ls command and write information concerning events in sets 5, 2, and 9 from hardware counters using the counter multiplexing mode, enter:
  - hpmcount -s 5,2,9 ls
Accessing gaussian

[Image of PuTTY Configuration window]

Category:
- Session
  - Logging
  - Terminal
  - Keyboard
  - Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - Telnet
  - Rlogin
  - SSH
  - Serial

Basic options for your PuTTY session

Specify the destination you want to connect to
Host Name (or IP address): gaussian.msi.umn.edu
Port: 22
Connection type:
- Raw
- Telnet
- Rlogin
- SSH
- Serial

Load, save or delete a stored session
Saved Sessions
- Default Settings
- austin_sage
- msi_calhoun
- msi_gaussian
- msi_116
- msi_15
- msi_silver

Close window on exit:
- Always
- Never
- Only on clean exit

About
Open
Cancel
#! /bin/csh
# Very simple serial code set up to execute under HPMCOUNT control.
cat << 'EOF' > ./it.f
  program main
  implicit none
  integer i
  real sum
  common sum
  sum=0.0
  do i=1,1000000
    sum=sum+exp(.00000001*i)
  end do
  print*, 'sum=', sum
  stop
end
'EOF'

# Compile and build program "it" from it.f, use -g option and no
# optimization to support source debugging of all Fortan statements:
xlf_r -O4 -qarch=auto -qrealsize=8 -o it it.f

# Execute program "it" with HPMCOUNT:
/usr/bin/hpmcount ./it

http://www.cisl.ucar.edu/docs/ibm/hpm.toolkit/hpmcount.html
Lab 3-2: hpmcout output

HPMCOUNT output:

Execution time (wall clock time): 0.057595 seconds

############################ Resource Usage Statistics ############################

Total amount of time in user mode : 0.015934 seconds
Total amount of time in system mode : 0.003379 seconds
Maximum resident set size : 8532 Kbytes
Average shared memory use in text segment : 0 Kbytes*sec
Average unshared memory use in data segment : 77 Kbytes*sec
Number of page faults without I/O activity : 2073
Number of page faults with I/O activity : 2
Number of times process was swapped out : 0
Number of times file system performed INPUT : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent : 0
Number of IPC messages received : 0
Number of signals delivered : 0
Number of voluntary context switches : 13
Number of involuntary context switches : 3

http://www.cisl.ucar.edu/docs/ibm/hpm.toolkit/hpmcount.html
Lab 3-3: hpmcout output

Set: 1
Counting duration: 0.019886103 seconds
PM_FPU_1FLOP (FPU executed one flop instruction) : 4000225
PM_FPU_FMA (FPU executed multiply-add instruction) : 11000076
PM_FPU_FSQRT_FDIV (FPU executed FSQRT or FDIV instruction) : 0
PM_CYC (Processor cycles) : 26428653
PM_RUN_INST_CMPL (Run instructions completed) : 47657875
PM_RUN_CYC (Run cycles) : 93529315

Utilization rate : 9.755 %
Flop : 26,000 Mflop
Flop rate (flops / WCT) : 451.435 Mflop/s
Flops / user time : 4627.772 Mflop/s
FMA percentage : 146.665 %

http://www.cisl.ucar.edu/docs/ibm/hpm.toolkit/hpmcount.html
• **Libhpm:**
  - Provides instrumented programs with a summary output for each instrumented region in a program.
  - This library supports serial and parallel (Message Passing Interface (MPI), threaded, and mixed mode) applications, written in Fortran, C, and C++.
  - Provides a programming interface to start and stop performance counting for an application program.
  - The part of the application program between the start and stop of performance counting is called an *instrumentation section*.
  - Any such instrumentation section is assigned a unique integer number as a section identifier.
Libhpm: Template
Biomedical Informatics & Computational Biology

```
hpmInit( tasked, "my program" );
hpmStart( 1, "outer call" );
do_work();
  hpmStart( 2, "computing meaning of life" );
  do_more_work();
  hpmStop( 2 );
hpmStop( 1 );
hpmTerminate( taskID )
```

- Calls to hpmInit() and hpmTerminate() embrace the instrumented part.
- Every instrumentation section starts with hpmStart() and ends with hpmStop().
- The section identifier is the first parameter to the latter two functions.
• The hardware performance counters information is the value of special CPU registers that are incremented at certain events

• The number of such registers is different for each architecture
# Registers per Architecture

<table>
<thead>
<tr>
<th>Processor Architecture</th>
<th>Number of Performance Counter Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power PC 970</td>
<td>8</td>
</tr>
<tr>
<td>POWER4</td>
<td>8</td>
</tr>
<tr>
<td>POWER5</td>
<td>8</td>
</tr>
<tr>
<td>POWER5+</td>
<td>6</td>
</tr>
<tr>
<td>POWER6</td>
<td>6</td>
</tr>
<tr>
<td>Blue Gene/L</td>
<td>52</td>
</tr>
<tr>
<td>Blue Gene/P</td>
<td>256</td>
</tr>
</tbody>
</table>
Counting Registers

• User sees private counter values for the application
• Counting of the special CPU registers is frozen, and the values are saved whenever the application process is taken off the CPU and another process is scheduled
• Counting is resumed when the user application is scheduled on the CPU
• The special CPU registers can count different events
• There are restrictions on which registers can count which events
### Performance Monitor Counters

<table>
<thead>
<tr>
<th>Processor</th>
<th>Performance Monitor Counters</th>
<th>Events</th>
<th>Event Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerPC 970</td>
<td>8</td>
<td>230</td>
<td>49</td>
</tr>
<tr>
<td>PowerPC 970 MP</td>
<td>8</td>
<td>230</td>
<td>51</td>
</tr>
<tr>
<td>POWER4</td>
<td>8</td>
<td>244</td>
<td>63</td>
</tr>
<tr>
<td>POWER4 II</td>
<td>8</td>
<td>244</td>
<td>63</td>
</tr>
<tr>
<td>POWER5</td>
<td>6</td>
<td>474</td>
<td>163</td>
</tr>
<tr>
<td>POWER5 II</td>
<td>6</td>
<td>483</td>
<td>188</td>
</tr>
<tr>
<td>POWER6</td>
<td>6</td>
<td>553</td>
<td>202</td>
</tr>
</tbody>
</table>
• What is performance tuning?
• Speedup and efficiency
• Silver an IBM JS22
• Compiling with the XL compilers
• **Identifying Performance Bottlenecks: MPI Tracer**
  - MPI tracer
    • Lab 4
Motivation: Message Passing Model

Task 0

Task 1

Task 2

Task 3

Task: a program with local memory and I/O ports

Channel: a message queue that connects two tasks

Computation + Communication
The MPI profiling and tracing library collects profiling and tracing data for MPI programs.

<table>
<thead>
<tr>
<th>Library name</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>libmpitrace.a</td>
<td>Library for both the C and Fortran applications</td>
</tr>
<tr>
<td>mpt.h</td>
<td>Header files</td>
</tr>
</tbody>
</table>
To use the library, the application must be compiled with the `-g` option
- You might consider turning off or having a lower level of optimization (`-O2, -O1,...`) for the application when linking with the MPI profiling and tracing library
- High level optimization affects the correctness of the debugging information and can also affect the call stack behavior

To link the application with the library:
- `-L/path/to/libraries`, where `/path/to/libraries` is the path where the libraries are located
- `-lmpitrace`, which should be before the MPI library `-lmptich`, in the linking order
- The option `-llicense` to link the license library
• **C example**

CC = /usr/lpp/ppe.poe/bin/mpcc_r  
TRACE_LIB = -L<path/to/libmpitrace.a> -lmpitrace  

mpitrace.ppe: mpi_test.c  

$(CC) -g -o $@ $< $(TRACE_LIB) -lm  

• **Fortran example**

FC = /usr/lpp/ppe.poe/bin/mpxlf_r  
TRACE_LIB = -L<path/to/libmpitrace.a> -lmpitrace  

swim.ppe: swim.f  

$(FC) -g -o $@ $< $(TRACE_LIB)
• **C example**

```
CC = /opt/ibmhpc/ppe.poe/bin/mpcc
TRACE_LIB = -L</path/to/libmpitrace.a> -lmpitrace

mpitrace: mpi_test.c
$(CC) -g -o @$ $(TRACE_LIB) -lm
```

• **Fortran example**

```
FC = /opt/ibmhpc/ppe.poe/bin/mpfort
TRACE_LIB = -L</path/to/libmpitrace.a> -lmpitrace

statusesf_trace: statusesf.f
$(FC) -g -o @$ $(TRACE_LIB)
```
Wrappers can save a record of all MPI events one after MPI Init(), until the application completes or until the trace buffer is full.
Tracing All Events: Finer Granularity

- Control the time-history measurement within the application by calling routines to start or stop tracing
  - Fortran syntax
    ```fortran
    call trace_start()
    do work + mpi ...
    call trace_stop()
    ```
  - C syntax
    ```c
    void trace_start(void);
    void trace_start(void);
    trace_start();
    do work + mpi ...
    trace_stop();
    ```
  - C++ syntax
    ```cpp
    extern "C" void trace_start(void);
    extern "C" void trace_start(void);
    trace_start();
    do work + mpi ...
    trace_stop();
    ```
• To use one of the previous control methods, the `TRACE_ALL_EVENTS` variable must be Disabled. Otherwise, it traces all events.

• You can use one of the following commands, depending on your shell, to disable the variable:

  bash
  export TRACE_ALL_EVENTS=no

  csh
  setenv TRACE_ALL_EVENTS no (csh)
Environmental Variables

- **TRACE_ALL_TASKS**
  - When saving MPI event records, it is easy to generate trace files that are too large to visualize. To reduce the data volume, when you set `TRACE_ALL_EVENTS=yes`

- **TRACE_MAX_RANK**
  - To provide more control, you can set `MAX_TRACE_RANK=#`
Environmental Variables - 2

- **TRACEBACK_LEVEL**
  - In some cases, there might be deeply nested layers on top of MPI and you might need to profile higher up the call chain (functions in the call stack). You can do this by setting this environment variable (default value is 0). For example, setting TRACEBACK_LEVEL=1 indicates that the library must save addresses starting with the parent in the call chain (level = 1), not with the location of the MPI call (level = 0).

- **SWAP_BYTES**
  - The event trace file is binary, and therefore, it is sensitive to byte order. For example, Blue Gene/L is big endian, and your visualization workstation is probably little endian (for example, x86). The trace files are written in little endian format by default. If you use a big endian system for graphical display, such as Apple OS/X, AIX on the System p workstation, and so on, you can set an environment variable by using one of the following commands depending on you shell:

  ```bash
  export SWAP_BYTES=no
  ```

  ```csh
  setenv SWAP_BYTES no
  ```

  Setting this variable results in a trace file in big endian format when you run your job.
In either profiling or tracing mode, there is an option to collect information about the number of hops for point-to-point communication on the torus network. This feature can be enabled by setting the TRACE_SEND_PATTERN environment variable as follows depending on your shell:

```bash
export TRACE_SEND_PATTERN=yes
```

```csh
setenv TRACE_SEND_PATTERN yes
```

Wrappers keep track of the number of bytes that are sent to each task, and a binary file `send_bytes.matrix` is written during MPI Finalize, which lists the number of bytes that were sent from each task to all other tasks. The binary file has the following format:

\[ D_{00}, D_{01}, ..., D_{0n}, D_{10}, ..., D_{ij}, ..., D_{nn} \]

In this format, the data type \( D_{ij} \) is double (in C), and it represents the size of MPI data that is sent from rank \( i \) to rank \( j \). This matrix can be used as input to external utilities that can generate efficient mappings of MPI tasks onto torus coordinates. The wrappers also provide the average number of hops for all flavors of MPI Send. The wrappers do not track the message-traffic patterns in collective calls, such as MPI Alltoall. Only point-to-point send operations are tracked. AverageHops for all communications on a given processor is measured as follows:

\[
\text{AverageHops} = \frac{\text{sum}(\text{Hops}_i \times \text{Bytes}_i)}{\text{sum}(\text{Bytes}_i)}
\]

\( \text{Hops}_i \) is the distance between the processors for MPI communication, and \( \text{Bytes}_i \) is the size of the data that is transferred in this communication. The logical concept behind this performance metric is to measure how far each byte has to travel for the communication (in average). If the communication processor pair is close to each other in the coordinate, the AverageHops value tends to be small.
• `mpi profile.taskid` has the timing summaries
• `mpi profile.0` file contains a timing summary from each task. Currently, for scalability reasons, only four ranks, rank 0 and rank with (min,med,max) MPI communication time, generate a plain text file by default
• To change this default setting, one simple function can be implemented and linked into compilation:

```
control.c:
int MT_output_trace(int rank) {
    return 1;
}
mpitrace: mpi_test.c
$(CC) $(CFLAGS) control.o mpi_test.o $(TRACE_LIB) -lm -o $@
```
mpi profile.0
elapsed time from clock-cycles using freq = 700.0 MHz

-----------------------------

MPI Routine #calls avg. bytes time(sec)
-----------------------------

MPI_Comp_size 1 0.0 0.000
MPI_Comp_rank 1 0.0 0.000
MPI_Isend 21 99864.3 0.000
MPI_Irecv 21 99864.3 0.000
MPI_Waitall 21 0.0 0.014
MPI_Barrier 47 0.0 0.000

-----------------------------
total communication time = 0.015 seconds.
total elapsed time = 4.039 seconds.
-----------------------------
Message size distributions:
MPI_Isend #calls avg. bytes time(sec)
3  2.3  0.000
1  8.0  0.000
1  16.0  0.000
1  32.0  0.000
1  64.0  0.000
1  128.0  0.000
1  256.0  0.000
1  512.0  0.000
1  1024.0  0.000
1  2048.0  0.000
1  4096.0  0.000
1  8192.0  0.000
1  16384.0  0.000
1  32768.0  0.000
1  65536.0  0.000
1  131072.0  0.000
1  262144.0  0.000
1  524288.0  0.000
1 1048576.0  0.000
Message size distributions:
MPI_Irecv #calls avg. bytes time(sec)
3  2.3  0.000
1  8.0  0.000
1 16.0  0.000
1 32.0  0.000
1 64.0  0.000
1 128.0  0.000
1 256.0  0.000
1 512.0  0.000
1 1024.0  0.000
1 2048.0  0.000
1 4096.0  0.000
1 8192.0  0.000
1 16384.0  0.000
1 32768.0  0.000
1 65536.0  0.000
1 131072.0  0.000
1 262144.0  0.000
1 524288.0  0.000
1 1048576.0  0.000

Communication summary for all tasks:
minimum communication time = 0.015 sec for task 0
median communication time = 4.039 sec for task 20
maximum communication time = 4.039 sec for task 30
<table>
<thead>
<tr>
<th>taskid</th>
<th>xcoord</th>
<th>ycoord</th>
<th>zcoord</th>
<th>procid</th>
<th>total_comm (sec)</th>
<th>avg_hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.015</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>7.00</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>23</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>7.00</td>
</tr>
<tr>
<td>taskid</td>
<td>xcoord</td>
<td>ycoord</td>
<td>zcoord</td>
<td>procid</td>
<td>total_comm (sec)</td>
<td>avg_hops</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.015</td>
<td>1.00</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>23</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>7.00</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>31</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>7.00</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>27</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>4.00</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4.039</td>
<td>1.00</td>
</tr>
</tbody>
</table>
cd /scratch1/cpsosa/bicb8510/c/mpi

module load hpct

silver> make -f make.pi
/opt/ibmhpc/ppe.poe/bin/mpcc -g -o pi pi.c -L/opt/ibmhpc/ppe.hpct/lib -lmpitrace -lm

silver> poe ./pi -hfile hostfile -procs 4
20
Enter the number of intervals: (0 quits) pi is approximately 3.1418009868930938, Error is 0.0002083333033007

0
Enter the number of intervals: (0 quits) wrote trace file: single_trace
Appendix I: Xprofiler Options

Biomedical Informatics & Computational Biology
• -b Xprofiler -b a.out gmon.out
  - This option suppresses the printing of the field descriptions for the
    Flat Profile, Call Graph Profile, and Function Index reports when
    they are written to a file with the Save As option of the File menu
• -s Xprofiler -s a.out gmon.out.1 gmon.out.2 gmon.out.3
  - If multiple gmon.out files are specified when Xprofiler is started,
    this option produces the gmon.sum profile data file. The gmon.sum
    file represents the sum of the profile information in all the
    specified profile files. Note that if you specify a single gmon.out
    file, the gmon.sum file contains the same data as the gmon.out file
• -z Xprofiler -z a.out gmon.out
  - This option includes functions that have both zero CPU usage and no
    call counts in the Flat Profile, Call Graph Profile, and Function Index
    reports. A function will not have a call count if the file that contains
    its definition was not compiled with the -pg option, which is common
    with system library files
• \(-a\) Xprofiler  
  \(-a\) pathA:@:pathB
  - This option adds alternative paths to search for source code and library files, or changes the current path search order. When using this command line option, you can use the at sign (@) to represent the default file path, in order to specify that other paths be searched before the default path.

• \(-c\) Xprofiler a.out gmon.out  
  \(-c\) config_file_name
  - This option loads the specified configuration file. If the \(-c\) option is used on the command line, the configuration file name specified with it is displayed in the Configuration File (-c): text field, in the Loads Files window, and the Selection field of the Load Configuration File window. When both the \(-c\) and \(-disp\_max\) options are specified on the command line, the \(-disp\_max\) option is ignored. However, the value that was specified with it is displayed in the Initial Display (-disp_max): field in the Load Files window the next time it is opened.

• \(-disp\_max\) Xprofiler  
  \(-disp\_max\) 50 a.out gmon.out
  - This option sets the number of function boxes that Xprofiler initially displays in the function call tree. The value that is supplied with this flag can be any integer between 0 and 5,000. Xprofiler displays the function boxes for the most CPU-intensive functions through the number that you specify. For instance, if you specify 50, Xprofiler displays the function boxes for the 50 functions in your program that consume the most CPU. After this, you can change the number of function boxes that are displayed via the Filter menu options. This flag has no effect on the content of any of the Xprofiler reports.
• `-e Xprofiler -e function1 -e function2 a.out gmon.out`
  This option de-emphasizes the general appearance of the function box or boxes for the specified function or functions in the function call tree. This option also limits the number of entries for these function in the Call Graph Profile report. This also applies to the specified function's descendants, as long as they have not been called by non-specified functions. In the function call tree, the function box or boxes for the specified function or functions appears to be unavailable. Its size and the content of the label remain the same. This also applies to descendant functions, as long as they have not been called by non-specified functions. In the Call Graph Profile report, an entry for the specified function only appears where it is a child of another function or as a parent of a function that also has at least one non-specified function as its parent. The information for this entry remains unchanged. Entries for descendants of the specified function do not appear unless they have been called by at least one non-specified function in the program.
-E Xprofiler -E function1 -E function2 a.out gmon.out

This option changes the general appearance and label information of the function box or boxes for the specified function or functions in the function call tree. In addition, this option limits the number of entries for these functions in the Call Graph Profile report and changes the CPU data that is associated with them. These results also apply to the specified function's descendants, as long as they have not been called by non-specified functions in the program. In the function call tree, the function box for the specified function appears to be unavailable, and its size and shape also change so that it appears as a square of the smallest allowable size. In addition, the CPU time shown in the function box label appears as zero. The same applies to function boxes for descendant functions, as long as they have not been called by non-specified functions. This option also causes the CPU time spent by the specified function to be deducted from the left side CPU total in the label of the function box for each of the specified ancestors of the function. In the Call Graph Profile report, an entry for the specified function only appears where it is a child of another function or as a parent of a function that also has at least one non-specified function as its parent. When this is the case, the time in the self and descendants columns for this entry is set to zero. In addition, the amount of time that was in the descendants column for the specified function is subtracted from the time listed under the descendants column for the profiled function. As a result, be aware that the value listed in the % time column for most profiled functions in this report will change.
-f Xprofiler -f function1 -f function2 a.out gmon.out

This option de-emphasizes the general appearance of all function boxes in the function call tree, except for that of the specified function or functions and its descendant or descendants. In addition, the number of entries in the Call Graph Profile report for the non-specified functions and non-descendant functions is limited. The -f flag overrides the -e flag. In the function call tree, all function boxes, except for that of the specified function or functions and its descendant or descendants, appear to be unavailable. The size of these boxes and the content of their labels remain the same. For the specified function or functions, and its descendant or descendants, the appearance of the function boxes and labels remains the same. In the Call Graph Profile report, an entry for a non-specified or non-descendant function only appears where it is a parent or child of a specified function or one of its descendants. All information for this entry remains the same.
• `-F` Xprofiler `–F` function1 `–F` function2 a.out gmon.out
  - This option changes the general appearance and label information of all function boxes in the function call tree, except for that of the specified function or functions and its descendants. In addition, the number of entries in the Call Graph Profile report for the non-specified and non-descendant functions is limited, and the CPU data associated with them is changed. The `-F` flag overrides the `-E` flag. In the function call tree, all function boxes, except for that of the specified function or functions and its descendant or descendants, appear to be unavailable. The size and shape of these boxes change so that they are displayed as squares of the smallest allowable size. In addition, the CPU time shown in the function box label appears as zero. In the Call Graph Profile report, an entry for a non-specified or non-descendant function only is displayed where it is a parent or child of a specified function or one of its descendants. When this is the case, the time in the self and descendants columns for this entry is set to zero. As a result, be aware that the value listed in the % time column for most profiled functions in this report will change.

• `-L` Xprofiler `–L` /lib/profiled
  - This option sets the path name for locating shared libraries. If you plan to specify multiple paths, use the Set File Search Paths option of the File menu on the Xprofiler GUI.
## Appendix II: Computer Performance

### Table of FLOPS

<table>
<thead>
<tr>
<th>Name</th>
<th>FLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>yotta FLOPS</td>
<td>$10^{24}$</td>
</tr>
<tr>
<td>zetta FLOPS</td>
<td>$10^{21}$</td>
</tr>
<tr>
<td>exa FLOPS</td>
<td>$10^{18}$</td>
</tr>
<tr>
<td>peta FLOPS</td>
<td>$10^{15}$</td>
</tr>
<tr>
<td>tera FLOPS</td>
<td>$10^{12}$</td>
</tr>
<tr>
<td>giga FLOPS</td>
<td>$10^9$</td>
</tr>
<tr>
<td>mega FLOPS</td>
<td>$10^6$</td>
</tr>
<tr>
<td>kilo FLOPS</td>
<td>$10^3$</td>
</tr>
</tbody>
</table>

Appendix III: Introduction to MSI
BICB Resource

Silver: IBM JS22 - QS22

Hardware and Configuration:
- 7 compute blades
- 1 interactive blade
- 1 file server/management node
- 30 total compute processors
- .72 TB total memory

Specifications for the compute blades are as follows:
- Six JS22 blades each with four 4.0 GHz Power6 processors and 8 GB of memory
- One QS22 blade with two 3.2 GHz PowerXCell 8i processors and 16 GB of memory

Specifications for the interactive blade are as follows:
- * One JS22 blade with four 4.0 GHz Power6 processor and 8 GB of memory

Network:
- All of the blades within the cluster are interconnected with a 4X InfiniBand DDR network.

https://www.msi.umn.edu/labs/umbcl/techinfo.html
MSI web pages

• The institute’s web page
  - https://www.msi.umn.edu

• Getting started
  - https://www.msi.umn.edu/support/start.html

• Software
  - https://www.msi.umn.edu/sw

• Password reset
  - https://www.msi.umn.edu/password

• Tutorials
  - https://www.msi.umn.edu/tutorial
Access MSI Linux machine from my Windows machine?

• You would need Putty and X-Win32 (or Xming) to connect to MSI Linux machines from your Windows machine. Putty provides a terminal window in which you can perform all your commands. X-Win32 allows you to transfer X session (the graphical display) from MSI Linux machine to your Windows machine.

• Here are the links for configuring them on your Windows machine:
  X-Win32 (http://static.msi.umn.edu/user_support/xclient/xwin_win.html)
  Putty (http://static.msi.umn.edu/user_support/xclient/xwin_config.html)

• To upload files from your Windows machine to a directory in the Linux machine, use WinSCP (http://static.msi.umn.edu/user_support/ssh/winscp_win.html) or any SFTP software.
The NX server/client system provides a modern, faster, and more reliable connection from your Windows, Mac or Linux machines to MSI’s constellation of Linux machines that span the various computational laboratories and core hardware. Unlike X11, NX transfers the complete desktop to your machine and provides a user experience much closer to that of working directly from an MSI Linux workstation.

The FreeNX server is now running on cardinal.msi.umn.edu.

The server details are
- Host = cardinal.msi.umn.edu
- Port = 22
- Desktop = Unix/KDE or Unix/Gnome

The NX client can be installed on your computer, if not already installed. You can find the client software for your operating system at http://www.nomachine.com/download.php.
Access MSI Linux machine from my Linux machine?

Use any terminal program and ssh with the following command:

```
ssh -X johndoe@L15.msi.umn.edu     or
ssh -Y jane@L15.msi.umn.edu
```

The machine name L15.msi.umn.edu can be replaced by any other machine name depending on the lab you are trying to connect. The username is optional if the machine from which you are connecting has the same user name as MSI username.
Access MSI Linux machine from my Mac?

If you are using a Mac, use X11 terminal program and `ssh` with the following command,

```
ssh -Y johndoe@L15.msi.umn.edu
```

The username is optional if the login from which you are connecting is same as the MSI username. The machine name `L15.msi.umn.edu` can be replaced by any other machine name depending on the lab you are trying to connect.
BMSDL Specific

The machines proline, alanine, and glycine can only be accessed locally. They are located on the left side of the room.

The machines vl1-vl7 can be accessed remotely.
For help

• By email
  help@msi.umn.edu

• Web
  www.msi.umn.edu

• Phone
  612-626-0802

• By appointment
References

- MSI-UMR BICB Computational Laboratory Technical Information: https://www.msi.umn.edu/labs/umbcl/techinfo.html