IBM Massively Parallel Blue Gene: Application Development

Carlos P Sosa
IBM and Biomedical Informatics & Computational Biology,
University of Minnesota Rochester
Rochester, Minnesota
Outline

Biomedical Informatics & Computational Biology

- Part I: Hardware
  - Historical perspective: Why do we need MPPs?
  - Overview of massively parallel processing (MPP)
  - Architecture
- Part II: Software
  - Overview
  - Compilers
  - MPI
  - Building and Running Examples on Blue Gene
    - Hands-on session 1
- Part III: Applications
  - MPP architecture and its impact on applications
  - Performance tools
  - Introduction to code optimization
    - Hands-on session 2
  - Mapping applications on a massively parallel architecture
  - Applications landscape
  - Challenges and characteristics of Life Sciences applications
  - Selected Bioinformatics applications
  - Selected Structural Biology applications
    - Hands-on session 3
- Summary

Biomedical Informatics & Computational Biology
Outline

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-n session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Summary

Biomedical Informatics & Computational Biology
Technological challenges

- The point to which we can shrink transistors has an absolute limit
- The shrinking of transistors yield difficult side effects (Electro-Magnetic Interference)
- Power leakage

Multi-processor shared-memory machines
- Fast, sophisticated interconnects with multiple-processors
Commodity computing

Large-scale machines could be achieved using individual CPUs networked, or clustered to function together as a single unit

Massively parallel processing (MPP) systems

From Kilobytes to Petabytes in 50 Years: http://www.eurekalert.org/features/doe/2002-03/dlnl-fkt062102.php
Supercomputer Peak Performance

BICB
Biomedical Informatics & Computational Biology

From Kilobytes to Petabytes in 50 Years: http://www.eurekalert.org/features/doe/2002-03/dlnl-fkt062102.php
Grand challenge* problems is a key part of high performance computing applications

Grand challenges are fundamental problems in science and engineering with broad economic and scientific impact, and whose solution can be advanced by applying high performance computing techniques and resources.
Different from the Rest

Biomedical Informatics & Computational Biology

Source: Pete Beckam, Director, ACLF, Argonne National Lab.
Pushing the Technology

Source: Pete Beckam, Director, ACLF, Argonne National Lab.
December 1999:

IBM Announces $100 Million Research Initiative to build World's Fastest Supercomputer

"Blue Gene" to Tackle Protein Folding Grand Challenge

YORKTOWN HEIGHTS, NY, December 6, 1999 -- IBM today announced a new $100 million exploratory research initiative to build a supercomputer 500 times more powerful than the world's fastest computers today. The new computer -- nicknamed "Blue Gene" by IBM researchers -- will be capable of more than one quadrillion operations per second (one petaflop). This level of performance will make Blue Gene 1,000 times more powerful than the Deep Blue machine that beat world chess champion Garry Kasparov in 1997, and about 2 million times more powerful than today's top desktop PCs.

Blue Gene's massive computing power will initially be used to model the folding of human proteins, making this fundamental study of biology the company's first computing "grand challenge" since the Deep Blue experiment. Learning more about how proteins fold is expected to give medical researchers better understanding of diseases, as well as potential cures.
- Limits of physical size (floor space)
- Power consumption
- Cooling needed to house and run the aggregated equipment
Design Considerations

- Widening gap between processor and DRAM clock rates
- Excessive heat generated by dense packaging and high switching frequency
- Disparity between processor clock rate and immediate vicinity peripheral devices (memory, I/O buses, etc.)
- Network performance

The speed of the processor is traded in favor of dense packaging and low power consumption per processor.
Blue Gene Technology Roadmap

Blue Gene/L
PPC 440 @ 700MHz
Scalable to 360+ TF

Blue Gene/P
PPC 450 @ 850MHz
Scalable to 3+ PF

Blue Gene/Q
Power Multi-Core
Scalable to 10+ PF

2004
2007
2011
Most Power, Space, and Cooling efficient Supercomputer

Published specs per peak performance

- Sun/Constellation
- Cray/XT4
- SGI/ICE
## Areas of Application

**Biomedical Informatics & Computational Biology**

- **Improve understanding** - significantly larger scale, more complex and higher resolution models; new science applications
- **Multiscale and multiphysics** - From atoms to mega-structures; coupled applications
- **Shorter time to solution** - Answers from months to minutes

### Areas of Application

- **Physics – Materials Science**
  - Molecular Dynamics
  - Geophysical Data Processing
  - Upstream Petroleum
- **Computation Fluid Dynamics**
- **Biological Modeling – Brain Science**
- **Life Sciences: In-Silico Trials, Drug Discovery**
- **Financial Modeling**
  - Streaming Data Analysis
- **Environment and Climate Modeling**
- **Life Sciences: Sequencing**
Outline

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Summary

Biomedical Informatics and Computational Biology (BICB)
How is BG/P Configured?

Service & Front End (Login) Nodes

1Gbe Service Network

SLES10
DB2
XLF
XLC/C++
GPFS
ENSSSL
TWS LL

Storage Subsystem

10Gbe Functional Network

File Servers

Blue Gene core rack
1024 Compute Nodes/rack
Up to 64 I/O Nodes/rack

Source: C. P. Sosa and B. Knutson, IBM System Blue Gene Solution: Blue Gene/P Application Development, SG24-7278-03 Redbooks, Draft Redbooks, last update 25 August 2009
IBM System Blue Gene/P®

**System-on-Chip (SoC)**
Quad PowerPC 450 w/ Double FPU
Memory Controller w/ ECC
L2/L3 Cache
DMA & PMU
Torus Network
Collective Network
Global Barrier Network
10GbE Control Network
JTAG Monitor

**SoC**
13.6 GF/s
8 MB EDRAM

**Compute Card**
1 SoC, 40 DRAMs
13.6 GF/s
2 GB DDR

**Node Card**
32 Compute Cards
0-2 I/O cards
435.2 GF/s
64 GB

**Rack**
32 Node Cards
13.9 TF/s
2 TB

**Cabled**
8x8x16

**System**
Up to 256 Racks
Up to 3.5 PF/s
Up to 512 TB

---

Hierarchy

Compute nodes dedicated to running user applications, and almost nothing else - simple compute node kernel (CNK)

I/O nodes run Linux and provide a more complete range of OS services - files, sockets, process launch, debugging, and termination

Service node performs system management services (e.g., heart beating, monitoring errors) - largely transparent to application/system software

Looking inside Blue Gene
Shared GPFS Filesystem

IBM p520
GPFS NSD
9 servers total

each server has one 10GbE
and one 8Gb FC connection

Force10 Networks
E1200

p520 NSDs

DDN Storage

Data Direct Networks
S2A9900 couplet
with 5 SAF6000 disk
enclosures attached

8 NSDs per couplet

*Note: not to scale, not all
connections are shown
BG/P Applications Specific Integrated Circuit (ASIC) Diagram

L2 Data cache: prefetch buffer holds 15 128-byte lines can prefetch up to 7 streams

L1 Data cache: 32 KB total size
- 32-Byte line size,
- 64-way associative
- round-robin replacement
- write-through for cache coherency
- 4-cycle load to use

L3 Data cache: 2x4 MB
~50 cycles latency on-chip

**Blue Gene/P Job Modes Allow Flexible Use of Node Memory**

### Virtual Node Mode
- Previously called Virtual Node Mode
- All four cores run one MPI process each
- No threading
- Memory / MPI process = \( \frac{1}{4} \) node memory
- MPI programming model

### Dual Node Mode
- Two cores run one MPI process each
- Each process may spawn one thread on core not used by other process
- Memory / MPI process = \( \frac{1}{2} \) node memory
- Hybrid MPI/OpenMP programming model

### SMP Node Mode
- One core runs one MPI process
- Process may spawn threads on each of the other cores
- Memory / MPI process = full node memory
- Hybrid MPI/OpenMP programming model
Blue Gene Integrated Networks

Biomedical Informatics & Computational Biology

- **Torus**
  - Interconnect to all compute nodes
  - Torus network is used
  - Point-to-point communication

- **Collective**
  - Interconnects compute and I/O nodes
  - One-to-all broadcast functionality
  - Reduction operations functionality

- **Barrier**
  - Compute and I/O nodes
  - Low latency barrier across system (< 1usec for 72 rack)
  - Used to synchronize timebases

- **10Gb Functional Ethernet**
  - I/O nodes only

- **1Gb Private Control Ethernet**
  - Provides JTAG, i2c, etc, access to hardware. Accessible only from Service Node system
  - Boot, monitoring, and diagnostics

- **Clock network**
  - Single clock source for all racks
High-Throughput Computing (HTC) modes on Blue Gene/P

- BG/P with HTC looks like a cluster for serial and parallel apps
- Hybrid environment ... standard HPC (MPI) apps plus now HTC apps
- Enables a new class of workloads that use many single-node jobs
- Easy administration using web-based Navigator
HPC versus HTC

High Performance Computing (HPC) Mode - best for Capability Computing
- Parallel, tightly coupled applications
  - Single Instruction, Multiple Data (SIMD) architecture
  - Programming model: typically MPI
- Apps need tremendous amount of computational power over short time period

High Throughput Computing (HTC) Mode - best for Capacity Computing
- Large number of independent tasks
  - Multiple Instruction, Multiple Data (MIMD) architecture
  - Programming model: non-MPI
- Apps need large amount of computational power over long time period
- Traditionally run on large clusters

HTC and HPC modes co-exist on Blue Gene
- Determined when resource pool (partition) is allocated
Outline

Biomedical Informatics & Computational Biology

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Summary

Biomedical Informatics and Computational Biology (BICB)
IBM Software Stack

- **XL (FORTRAN, C, and C++) compilers**
  - Externals preserved
  - Optimized for specific BG functions
  - OpenMP support

- **LoadLeveler scheduler**
  - Same externals for job submission and system query functions
  - Backfill scheduling to achieve maximum system utilization

- **GPFS parallel file system**
  - Provides high performance file access, as in current pSeries and xSeries clusters
  - Runs on I/O nodes and disk servers

- **ESSL/MASSV libraries**
  - Optimization library and intrinsics for better application performance
  - Serial Static Library supporting 32-bit applications
  - Callable from FORTRAN, C, and C++

- **MPI library**
  - Message passing interface library, based on MPICH2, tuned for the Blue Gene architecture

Other Software Support

- **Parallel File Systems**
  - Lustre at LLNL, PVFS2 at ANL

- **Job Schedulers**
  - SLURM at LLNL, Cobalt at ANL
  - Altair PBS Pro, Platform LSF (for BG/L only)
  - Condor HTC (porting for BG/P)

- **Parallel Debugger**
  - Etnus TotalView (for BG/L as of now, porting for BG/P)
  - Allinea DDT and OPT (porting for BG/P)

- **Libraries**
  - FFT Library - Tuned functions by TU-Vienna
  - VNI (porting for BG/P)

- **Performance Tools**
  - HPC Toolkit: MP_Profiler, Xprofiler, HPM, PeekPerf, PAPI
  - Tau, Paraver, Kojak
Theoretical floating-point performance

- 1 fpmadd per cycle
- Total of 4 floating-point operations per cycle
- 4 floating-point operations/cycle x 850 cycle/s x 10^6 = 3,400 x 10^6 = 3.4 GFlop/s per core
- Peak performance = 13.6 GFlop/s per node (4 cores)
## Two Generations of Blue Gene

Biomedical Informatics & Computational Biology

### Feature Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Blue Gene/L</th>
<th>Blue Gene/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores per node</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Core Clock Speed</td>
<td>700 MHz</td>
<td>850 MHz</td>
</tr>
<tr>
<td>Cache Coherency</td>
<td>Software managed</td>
<td>SMP</td>
</tr>
<tr>
<td>Private L1 cache</td>
<td>32 KB per core</td>
<td>32 KB per core</td>
</tr>
<tr>
<td>Private L2 cache</td>
<td>14 stream prefetching</td>
<td>14 stream prefetching</td>
</tr>
<tr>
<td>Shared L3 cache</td>
<td>4 MB</td>
<td>8 MB</td>
</tr>
<tr>
<td>Physical Memory per Node</td>
<td>512 MB - 1 GB</td>
<td>2 GB</td>
</tr>
<tr>
<td>Main Memory Bandwidth</td>
<td>5.6 GB/s</td>
<td>13.6 GB/s</td>
</tr>
<tr>
<td>Peak Performance</td>
<td>5.6 GFlop/s per node</td>
<td>13.6 GFlop/s per node</td>
</tr>
</tbody>
</table>

### Full System (72 rack comparison)

<table>
<thead>
<tr>
<th>Feature</th>
<th>Blue Gene/L</th>
<th>Blue Gene/P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Performance</td>
<td>410 TFlop/s</td>
<td>~1 PFlop/s</td>
</tr>
<tr>
<td>Power</td>
<td>1.7 MW</td>
<td>~2.3 MW</td>
</tr>
</tbody>
</table>

- 2048 MB or 4096 memory per node, 32-bit memory addressing

- Compute-node kernel does not have full Linux (limited system calls) compatibility
  - no fork() or system() calls
Compilers for Blue Gene are located in the front-end (/opt/ibmcmp)

- **Fortran:**
  - /opt/ibmcmp/xlf/bg/11.1/bin/bgxlf
  - /opt/ibmcmp/xlf/bg/11.1/bin/bgxlf90
  - /opt/ibmcmp/xlf/bg/11.1/bin/bgxlf95

- **C:**
  - /opt/ibmcmp/vac/bg/9.0/bin/bgxlc

- **C++:**
  - /opt/ibmcmp/vacpp/bg/9.0/bin/bgxlc
- **C:** bgc89, bgc99, bgcc, bgxlc bgc89_r, bgc99_r bgcc_r, bgxlc_r
- **C++:** bgxlc++, bgxlc++_r, bgxlc, bgxlc_r
- **Fortran:** bgf2003, bgf95, bgxlf2003, bgxlf90_r, bgxlf_r, bgf77, bgfort77, bgxlf2003_r, bgxlf95, bgf90, bgxlf, bgxlf90, bgxlf95_r
The following compiler options, although available for other IBM systems, are not supported by the Blue Gene/P hardware:

- **-q64**: The Blue Gene/P system uses a 32-bit architecture; you cannot compile in 64-bit mode.

- **-qaltivec**: The 450 processor does not support VMX instructions or vector data types.
The Standard GNU compilers and libraries which are also located on the frontend node will NOT produce Blue Gene compatible binary code. The standard GNU compilers can only be used for utility or frontend code development that your application may require.

GNU compilers (Fortran, C, C++) for Blue Gene are located in (/opt/blrts-gnu/)

- **Fortran:**
  - `/opt/gnu/powerpc-bgp-linux-gfortran`

- **C:**
  - `/opt/gnu/powerpc-bgp-linux-gcc`

- **C++:**
  - `/opt/gnu/powerpc-bgp-linux-g++`

It is recommended not to use GNU compiler for Blue Gene as the IBM XL compilers offer significantly higher performance. The GNU compilers do offer more flexible support for things like inline assembler.
Messing Software Stack

MPI implementation on Blue Gene is based on MPICH-2 from Argonne National Laboratory.

Include files `mpi.h` and `mpif.h` are at the location:

- `/bgsys/drivers/ppcfloor/comm/include`
The following scripts are provided to compile and link MPI programs:

- **mpicc** C compiler
- **mpicxx** C++ compiler
- **mpif77** Fortran 77 compiler
- **mpif90** Fortran 90 compiler
- **mpixlc** IBM XL C compiler
- **mpixlc_r** Thread-safe version of mpixlc
- **mpixlcxx** IBM XL C++ compiler
- **mpixlcxx_r** Thread-safe version of mpixlcxx
- **mpixlf2003** IBM XL Fortran 2003 compiler
- **mpixlf2003_r** Thread-safe version of mpixlf2003
- **mpixlf77** IBM XL Fortran 77 compiler
- **mpixlf77_r** Thread-safe version of mpixlf77
- **mpixlf90** IBM XL Fortran 90 compiler
- **mpixlf90_r** Thread-safe version of mpixlf90
- **mpixlf95** IBM XL Fortran 95 compiler
- **mpixlf95_r** Thread-safe version of mpixlf95
- **mpich2version** Prints MPICH2 version information
$ make -f make.hello
$ mpixlc_r -O3 -qarch=450 -qtune=450 hello.c -o hello

$cat make.hello
XL_CC       =  mpixlc_r
OBJ         =  hello
SRC         =  hello.c
FLAGS       =  -O3 -qarch=450  -qtune=450
LIBS        =  $(OBJ):  $(SRC)

$(OBJ):  $(SRC)
     ${XL_CC}  $(FLAGS)  $(SRC)  -o  $(OBJ)  $(LIBS)

clean:
     rm  *.o  hello
Hello World: C

Hello World: C

$ cat hello.c
#include <stdio.h>      /* Headers */
#include "mpi.h"

main(int argc, char **argv)  /* Function main */
{
    int rank, size, tag, rc, i;
    MPI_Status status;
    char message[20];

    rc = MPI_Init(&argc, &argv);
    rc = MPI_Comm_size(MPI_COMM_WORLD, &size);
    rc = MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    tag = 100;

    if(rank == 0) {
        strcpy(message, "Hello, world");
        for (i=1; i<size; i++)
            rc = MPI_Send(message, 13, MPI_CHAR, i, tag, MPI_COMM_WORLD);
    } else
        rc = MPI_Recv(message, 13, MPI_CHAR, 0, tag, MPI_COMM_WORLD, &status);

    printf( "node %d : %.13s\n", rank, message);
    rc = MPI_Finalize();
}
$ cat make.hello

XL_CC       =  mpixlcxx_r
OBJ         =  hello
SRC         =  hello.cc
FLAGS       =  -O3 -qarch=450 -qtune=450
LIBS        =

$(OBJ):  $(SRC)
  ${XL_CC} $(FLAGS) $(SRC) -o $(OBJ)
$(LIBS)

clean:
  rm *.o hello
```cpp
// Include the MPI version 2 C++ bindings:
#include <mpi.h>
#include <iostream>
#include <string.h>
using namespace std;

int main(int argc, char* argv[])
{
    MPI::Init(argc, argv);

    int rank = MPI::COMM_WORLD.Get_rank();
    int size = MPI::COMM_WORLD.Get_size();

    char name[MPI_MAX_PROCESSOR_NAME];
    int len;
    memset(name, 0, MPI_MAX_PROCESSOR_NAME);
    MPI::Get_processor_name(name, len);
    memset(name+len, 0, MPI_MAX_PROCESSOR_NAME-len);

    cout << "hello_parallel.cc: Number of tasks=\"<<size<<\" My rank=\" << rank << \" My name=\"<<name<<\".\"<<endl;

    MPI::Finalize();
    return 0;
}
```
c Fortran example

program hello

include 'mpif.h'

integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)

call MPI_INIT(ierror)
call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
print*, 'node', rank, ': Hello world'
call MPI_FINALIZE(ierr)
end
The Compute Node Kernel, which provides the low-level primitives that are necessary to debug an application.

- The control and I/O daemon (CIOD) running on the I/O Nodes, which provides control and communications to Compute Nodes.

- A “debug server” running on the I/O Nodes, which is vendor-supplied code that interfaces with the CIOD.

- A debug client running on a Front End Node, which is where the user does their work interactively.
  - GNU Project debugger
  - Core processor debugger
  - Addr2Line utility
Outline

Biomedical Informatics & Computational Biology

- Part I: Hardware
  - Historical perspective: Why do we need MPPs?
  - Overview of massively parallel processing (MPP)
  - Architecture

- Part II: Software
  - Overview
  - Compilers
  - MPI
  - Building and Running Examples on Blue Gene
    - Hands-on session 1

- Part III: Applications
  - MPP architecture and its impact on applications
  - Performance tools
  - Introduction to code optimization
    - Hands-on session 2
  - Mapping applications on a massively parallel architecture
  - Applications landscape
  - Challenges and characteristics of Life Sciences applications
  - Selected Bioinformatics applications
  - Selected Structural Biology applications
    - Hands-on session 3

- Future Directions

- Summary
## Cards Naming Convention

### Service Cards:
- `Rxx-Mx-S` Service Card
  - Midplane (0-1) 0=Bottom, 1=Top
  - Rack Column (0-F)
  - Rack Row (0-F)
- Note: Master service card for a rack is always `Rxx-M0-S`

### Compute Cards:
- `Rxx-Mx-Nxx-Jxx` Compute Card (04 through 35)
  - Node Card (00-15)
  - Midplane (0-1) 0=Bottom, 1=Top
  - Rack Column (0-F)
  - Rack Row (0-F)

### Link Cards:
- `Rxx-Mx-Lx` Link Card (0-3)
  - Midplane (0-1) 0=Bottom, 1=Top
  - Rack Column (0-F)
  - Rack Row (0-F)

### I/O Cards:
- `Rxx-Mx-Nxx-Jxx` I/O Card (00-01)
  - Node Card (00-15)
  - Midplane (0-1) 0=Bottom, 1=Top
  - Rack Column (0-F)
  - Rack Row (0-F)

### Node Cards:
- `Rxx-Mx-Nxx` Node Card (00-15)
  - 0=Bottom Front, 1=Top Front
  - 0=Bottom Rear, 1=Top Rear

### Table - Cards Naming Convention

<table>
<thead>
<tr>
<th>Card</th>
<th>Element</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>Card</td>
<td>J04 through J35</td>
<td>R23-M10-N02-J09</td>
</tr>
<tr>
<td>I/O</td>
<td>Card</td>
<td>J00 through J01</td>
<td>R57-M1-N04-J00</td>
</tr>
<tr>
<td>I/O &amp; Compute</td>
<td>Module</td>
<td>U00</td>
<td>R23-M0-N13-J08-U00</td>
</tr>
<tr>
<td>Link</td>
<td>Module</td>
<td>U00 through U05 (00 left most, 05 right most)</td>
<td>R32-M0-L2_U03</td>
</tr>
<tr>
<td>Link</td>
<td>Port</td>
<td>TA through TF</td>
<td>R01-M0-L1-U02-TC</td>
</tr>
<tr>
<td>Link data cable</td>
<td>Connector</td>
<td>J00 through J15 (as labeled on link card)</td>
<td>R21-M1-L2-J13</td>
</tr>
<tr>
<td>Node Ethernet</td>
<td>Connector</td>
<td>EN0, EN1</td>
<td>R16-M1-N14-EN1</td>
</tr>
<tr>
<td>Service</td>
<td>Connector</td>
<td>Control FPGA, Control Network *, Clock R, Clock B</td>
<td>R05-M0-S-Control FPGA,</td>
</tr>
<tr>
<td>Clock</td>
<td>Connector</td>
<td>Input, Output 0 through Output 9</td>
<td>R13-K- Output 3</td>
</tr>
</tbody>
</table>
Submitting Jobs: mpirun

- **Job submission using mpirun**
  - User can use “mpirun” to submit jobs.
  - The Blue Gene mpirun is located in /usr/bin/mpirun

- **Typical use of mpirun:**
  - mpirun -np <# of processes> -partition <block id> -cwd `pwd` -exe <executable>

- **Where:**

- `-np`: Number of processors to be used. Must fit in available partition
- `-partition`: A partition from Blue Gene rack on which a given executable will execute, eg., R000.
- `-cwd`: The current working directory and is generally used to specify where any input and output files are located.
- `-exe`: The actual binary program which user wish to execute.

**Example:**

```bash
```
Comparison between `mpirun` and Loadleveler `llsubmit` command

<table>
<thead>
<tr>
<th>MPiRUN Syntax</th>
<th>LoadLeveler Command File Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mpirun --partition R011</code></td>
<td><code>#@ job_type = bluegene</code></td>
</tr>
<tr>
<td>--exe \</td>
<td><code>#@ requirements = (Machine == &quot;$\text{(host)\})\)\)</code></td>
</tr>
<tr>
<td>/gpfs/fs2/frontend_id/my_dir/my_prog \</td>
<td><code>#@ arguments = --exe \</code> \</td>
</tr>
<tr>
<td>--cwd \pwd`</td>
<td><code>    /gpfs/fs2/frontend_id/my\_dir/my\_prog</code> \</td>
</tr>
<tr>
<td>--args &quot;-f arg0 arg1&quot;</td>
<td><code>    -cwd \pwd\</code> --args &quot;-f arg0 arg1&quot;</td>
</tr>
<tr>
<td>2--verbose</td>
<td><code>#@ bg\_partition = R011</code></td>
</tr>
<tr>
<td></td>
<td><code>#@ queue</code></td>
</tr>
<tr>
<td></td>
<td><code>/bgsys/drivers/ppcfloor/bin/mpirun</code></td>
</tr>
</tbody>
</table>

`job_type` and `requirements` tags must **ALWAYS** be specified as listed above.

If the above command file listing were contained in a file named `my_job.cmd`, then the job would then be submitted to the LoadLeveler queue using `llsubmit my_job.cmd`. 
Outline

Biomedical Informatics & Computational Biology

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Future Directions

Summary
“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 Challenge?

Applications ...

... are we there?
Answer the following questions to help you in the decision-making process of porting applications and the level of effort required (answering “yes” to most of the questions is an indication that your code is already enabled for distributed-memory systems and a good candidate for Blue Gene/P):

1. Is the code already running in parallel?
2. Is the application addressing 32-bit?
3. Does the application rely on system calls, for example, `system`?
4. Does the code use the Message Passing Interface (MPI), specifically MPICH? Of the several parallel programming APIs, the only one supported on the Blue Gene/P system that is portable is MPICH. OpenMP is supported only on individual nodes.
5. Is the memory requirement per MPI task less than 4 GB?
6. Is the code computational intensive? That is, is there a small amount of I/O compared to computation?
7. Is the code floating-point intensive? This allows the double floating-point capability of the Blue Gene/P system to be exploited.
8. Does the algorithm allow for distributing the work to a large number of nodes?
9. Have you ensured that the code does not use flex lm licensing? At present, flex lm library support for Linux on IBM System p® is not available.

If you have answered “yes” to all of these questions, then answer the following questions:

1. Has the code been ported to Linux on System p?
2. Is the code Open Source Software (OSS)? These type of applications require the use of the GNU standard `configure` and special considerations are required.
3. Can the problem size be increased with increased numbers of processors?
4. Do you use standard input? If yes, can this be changed to single file input?
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

Tasks → Work → Time
Application Optimization

Application performance analysis

Memory bound?

I/O bound?

CPU bound?
Optimization Steps

1. Tune for compiler optimization flags
2. Locate hot-spots in the code
3. Use highly tuned libraries MASS/ESSSL
4. Manually optimize the code
5. Determine if I/O plays a role and tune if needed
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 time</th>
<th>Parallel time</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

Programmer A
Programmer B
Programmer C

Processors

Time reduction

Time

100
90
80
70
60
50
40
30
20
10
0

1
4

Programmer A
Programmer B
Programmer C
Outline

Biomedical Informatics & Computational Biology

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Future Directions

Summary
High Performance Computing Toolkit
- Xprofiler for CPU profiling
- Hardware Performance Monitoring (HPM)
- Message Passing Interface (MPI) Profiler and Tracer tool
- Threading performance: OpenMP profiling
- I/O Performance

GUI of the High Performance Computing Toolkit (HPCT)
HPC Toolkit Flow

HPM  MPI  OpenMP  MIO

C / C++ /Fortran Binary

Output/Analysis
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

- **Important factors:**
  - Sampling interval is in the order of ms
  - Profiling introduces overhead due to function calls
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.
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.
o 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

o 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

o 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
Xprofiler - Zoom In
Biomedical Informatics & Computational Biology

Program: mander  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)
Looking at the Source Code

File: Utility

<table>
<thead>
<tr>
<th>line</th>
<th>no. ticks</th>
<th>source code</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td></td>
<td>DO 50, I = 1, M</td>
</tr>
<tr>
<td>231</td>
<td></td>
<td>C(I, J) = ZERO</td>
</tr>
<tr>
<td>232</td>
<td>50</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>233</td>
<td></td>
<td>ELSE IF(BETA, NE, ONE) THEN</td>
</tr>
<tr>
<td>234</td>
<td></td>
<td>DO 60, I = 1, M</td>
</tr>
<tr>
<td>235</td>
<td>1</td>
<td>C(I, J) = BETA*C(I, J)</td>
</tr>
<tr>
<td>236</td>
<td>60</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>237</td>
<td></td>
<td>END IF</td>
</tr>
<tr>
<td>238</td>
<td></td>
<td>DO 80, L = 1, K</td>
</tr>
<tr>
<td>239</td>
<td></td>
<td>IF(B(L, J), NE, ZERO) THEN</td>
</tr>
<tr>
<td>240</td>
<td></td>
<td>TEMP = ALPHA*B(L, J)</td>
</tr>
<tr>
<td>241</td>
<td>1</td>
<td>DO 70, I = 1, M</td>
</tr>
<tr>
<td>242</td>
<td>850</td>
<td>C(I, J) = C(I, J) + TEMP*A(I, L)</td>
</tr>
<tr>
<td>243</td>
<td>70</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>244</td>
<td></td>
<td>END IF</td>
</tr>
<tr>
<td>245</td>
<td>80</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>246</td>
<td>90</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>247</td>
<td>ELSE</td>
<td></td>
</tr>
<tr>
<td>248</td>
<td>*</td>
<td>Form C := alpha<em>A'B + beta</em>C</td>
</tr>
<tr>
<td>249</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Search Engine: (regular expressions supported)

\[ \text{dgemm} \]
Looking at Assembler Code
<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  calculation starts: loop over the data gathered in the temporary  
c  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:
c

delrinv = 1.0/sqrt(delr2)
delr = delr2*delrinv
delr2inv = delrinv*delrinv
x = dxdr*delr
ind = eedtbdns*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
Loop over the 12-6 LJ terms for eedmeth = 1

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
#ifdef 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
    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

<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
% cd/scratch1 cpsosa bicb8510 fortran dgemm

% module load xlf

% make -f make.ibml
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
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"
- **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 vs. 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
Extra logic inserted in the processor to count specific events

Updated at every cycle

Strengths:
- Non-intrusive
- Very accurate
- Low overhead

Weakness
- Provides only hard counts
- Specific for each processor
- Access is not well documented
- Lack of standard and documentation on what is counted
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
```
#!/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 Fortran 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
**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.
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.
<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>
- 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
<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>
HPM Metrics

- Derived metrics allow users to correlate the behavior of the application to one or more of the hardware components.
- One can define threshold values acceptable for metrics and take actions regarding program optimization when values are below the threshold.

**Useful derived metrics**

- Cycles
- Instructions
- Floating point instructions
- Integer instructions
- Load/stores
- Cache misses
- TLB misses
- Branch taken / not taken
- Branch mispredictions

- IPC - instructions per cycle
- Float point rate (Mflip/s)
- Computation intensity
- Instructions per load/store
- Load/stores per cache miss
- Cache hit rate
- Loads per load miss
- Stores per store miss
- Loads per TLB miss
- Branches mispredicted %
Motivation: Message Passing Model

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 -lmpich, 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
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**
  ```c++
  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)
```
- **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

- **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 your shell:

    bash
    ```bash
    export SWAP_BYTES=no
    ```

    csh
    ```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
echo export TRACE_SEND_PATTERN=yes
```

```csh
echo 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:

```
D00,D01, ...D0n,D10, ...,Dij, ...,Dnn
```

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:

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

Hops$_i$ is the distance between the processors for MPI communication, and 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)
------------------------------------------------------------------
<table>
<thead>
<tr>
<th>Routine</th>
<th>Calls</th>
<th>Avg Bytes</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Comm_size</td>
<td>1</td>
<td>0.0</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>1</td>
<td>0.0</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Isend</td>
<td>21</td>
<td>99864.3</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>21</td>
<td>99864.3</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>21</td>
<td>0.0</td>
<td>0.014</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>47</td>
<td>0.0</td>
<td>0.000</td>
</tr>
</tbody>
</table>
------------------------------------------------------------------
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:

<table>
<thead>
<tr>
<th>Message size</th>
<th>MPI_Irecv #calls</th>
<th>avg. bytes</th>
<th>time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2.3</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>8.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>16.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>32.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>64.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>128.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>256.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>512.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>1024.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>2048.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>4096.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>8192.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>16384.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>32768.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>65536.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>131072.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>262144.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>524288.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>1</td>
<td>1048576.0</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

---

### 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>
</tbody>
</table>
### mpi profile.0 - 5

Biomedical Informatics & Computational Biology

**MPI tasks sorted by communication time:**

<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>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>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>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
Xprofiler Options - 1

- **-b Xprofiler**
  - 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**
  - 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**
  - 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.
o **-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.

o **-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

#### Biomedical Informatics & Computational Biology

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

Outline

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Summary

Biomedical Informatics & Computational Biology
Scientific and Engineering Applications Landscape

- Discrete Event Simulation
- Monte Carlo Methods
- Graph Theory
- Pattern Matching
- Symbolic Processing
- N-Body Problems
- Partial Differential Equations
- Transport Phenomena
- Ordinary Differential Equations
- Fields
- Fluid Dynamics
- Structural Mechanics
- Weather and Climate
- Multiphase Flow

- Rational Drug Design
- Fracture Mechanics
- Materials Sciences
- Nanotechnology
- Stochastic Process
- VLSI Design
- Network Flow
- Elementary Flow

- Molecular Dynamics
- Molecular Modeling

- Molecular Modeling

- Bioinformatics
  - Data Mining
  - Genome Computing
  - Proteomics

- Cryptography
- Seismic Processing
- Aerodynamics
- Particle Physics
- Structural Display

- Scientific and Engineering Applications Landscape

- Biomedical Informatics & Computational Biology
MPP Challenges for Applications Developers

- MPP flops are not only dependent on the individual performance of the CPU
- Performance on the holistic system
  - Memory system
  - File access
  - Network (messaging)
  - This type of system is not appropriate for every application
    - It is harder to take advantage of all processors
- Applications that can take advantage of large number of processors need access to larger systems
New Hardware Architectures

Applications Enablement?

Develop New Software Package
- Multidisciplinary Team

Port and Optimize Existing Application
- Collaboration/Developers
**CHARACTERISTICS**

These models were developed to describe molecular structures and properties in as practical a manner as possible for very large systems mainly of biological interest.

**CHALLENGE**

- Micro-seconds scale simulations require an order of $10^3$ increases in the computing power of contemporary high-end systems.
- Improving code performance and scalability for longer time and length scales simulations.
- Novel algorithms:
  - To reduce the performance and scaling bottlenecks.
  - To minimize memory requirements for large systems.

**USAGE**

- Protein modeling: structure, folding, dynamics and function.
- Compute-intensive applications.
CHARACTERISTICS
Molecular docking is used in structure-based drug design. The computational aspects can be divided into two parts. Ligand atoms being located inside the cavity or binding pocket of a receptor, which is a large biomolecule and scoring or identifying the most favorable interactions.

CHALLENGE
- Improving code performance and scalability for virtual screening of millions of ligands

USAGE
- Drug Discovery
- Compute-intensive applications
CHARACTERISTICS

- Science necessary to manage, process, and understand large amounts of data, for instance from the sequencing of the human genome, or from large databases containing information about plants and animals.

CHALLENGE

- Database fragmentation and distribution
- Parallelization of very large databases versus very large queries

USAGE

- Database searches; homology
- Data-intensive applications
CHARACTERISTICS
- These methods have traditionally been used for computing very accurate properties of small molecules to
- Complex systems with 1000s of atoms

CHALLENGE
- Parallel scalability to large number of processors
- Parallelization of Linear Algebra based-algorithms

USAGE
- Small to medium molecules properties
- Compute-intensive applications
- HMMER
- mpiBLAST-PIO
- PBPI
Hidden Markov models (HMMs) were initially introduced for pattern recognition in digitized acoustics of the human voice

Hidden Markov Models in Bioinformatics

- The UC Santa Cruz profile HMM software (SAM), probably the closest relative of HMMER
- Philipp Bucher's PFTOOLS package implements "generalized profiles", which are substantially similar to profile HMMs
- The commercial HMMpro package from Pierre Baldi and Yves Chauvin at NetID, Inc. implements more general HMM architectures than just profile HMMs, and also comes with a nifty Java display
- Andy Neuwald's PROBE software implements models based on multiple ungapped HMM motifs, and includes an implementation of training models by Gibbs sampling
- The UC San Diego META-MEME package from Michael Gribskov, Bill Grundy, Tim Bailey, and others implements multiple ungapped HMM motif models, similar to PROBE
- NCBI's PSI-BLAST server implements a stripped down but ultra-fast version of iterative profile HMM searches. This is a convenient Web server for folks who don't want to hassle with installing software locally
- Ewan Birney's WISETOOLS package can take a HMMER model and search it against EST or genomic DNA sequence, doing six-frame translation and allowing for frameshifts and introns
Several software packages are currently available:

- **HMMER** [http://hmmer.janelia.org/](http://hmmer.janelia.org/)
- **SAM** [http://www.cse.ucsc.edu/research/compbio/sam.html](http://www.cse.ucsc.edu/research/compbio/sam.html)
- **PFTOOLS** [http://www.isrec.isb-sib.ch/profile/profile.html](http://www.isrec.isb-sib.ch/profile/profile.html)
- **GENEWISE** [http://www.ebi.ac.uk/Wise2/](http://www.ebi.ac.uk/Wise2/)
- **META-MEME** [http://metameme.sdsc.edu/](http://metameme.sdsc.edu/)
HMMs and Applications

- **HMM:**
  - Profile HMMs are statistical models of multiple sequence alignments
    - Capture position-specific information on how conserved each column of the alignment is, and which residues are likely

- **Applications:**
  - Evolutionary homology in family of proteins
  - Automated annotation of the domain structure of proteins
  - Automated construction and maintenance of large multiple alignment databases

Source: HMMER’s User Guide 2.3.2
- **hmmalign**  Align sequences to an existing model
- **hmmbuild**  Build a model from a multiple sequence alignment
- **hmmcalibrate**  Takes an HMM and empirically determines parameters that are used to make searches more sensitive, by calculating more accurate expectation value scores (E-values)
- **hmmconvert**  Convert a model file into different formats, including a compact HMMER 2 binary format, and “best effort” emulation of GCG profiles
- **hmmemit**  Emit sequences probabilistically from a profile HMM
- **hmmfetch**  Get a single model from an HMM database
- **hmmindex**  Index an HMM database
- **hmmpfam**  Search an HMM database for matches to a query sequence
- **hmmsearch**  Search a sequence database for matches to an HMM

Three have been parallelized:

- **hmmcalibrate**: Takes an HMM and empirically determines parameters that are used to make searches more sensitive, by calculating more accurate expectation value scores (E-values)

- **hmmpfam**: is used to search a profile HMM database to a sequence query

- **hmmsearch**: is used to carry out sequence database searches to match an HMM

---

Queries:

- The two queries consisted of gi|1174687|sp|p42461|THIX_CORGL Thiamine biosynthesis protein X and 50 aligned globin sequences as provided in the HMMER version 2.2 (globins50)

- The first query corresponds to a single sequence of a small protein with 135 characters (amino acids)

Databases:

- Small protein database SWISS-PROT; 49787460 characters or 108891 sequences.
- The second database NR; 459219939 characters 929420 sequences

- This second database is larger than the first one by almost a factor of 10
Profiler Partial Output

<table>
<thead>
<tr>
<th>index</th>
<th>%time</th>
<th>self descendents</th>
<th>called+total</th>
<th>name</th>
<th>index</th>
<th>children</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.57</td>
<td>765.52</td>
<td>1/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>89.4</td>
<td>2.57</td>
<td>765.52</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>714.02</td>
<td>13.71</td>
<td>727771/727771</td>
<td>P7Viterbi</td>
<td>[3]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.06</td>
<td>15.49</td>
<td>727772/727772</td>
<td>ReadSeq</td>
<td>[5]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.94</td>
<td>5.10</td>
<td>727771/749381</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.TraceScoreCorrection [9]</td>
<td></td>
<td></td>
<td>3.61</td>
<td>5.48</td>
<td>230545997/230546004</td>
<td>toupper [12]</td>
</tr>
<tr>
<td>0.17</td>
<td>1.45</td>
<td>4366626/13380941</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>0.65</td>
<td>3720/3720</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.PostprocessSignificantHit [41]</td>
<td></td>
<td></td>
<td>0.24</td>
<td>0.00</td>
<td>2183313/36337220</td>
<td>log [17]</td>
</tr>
<tr>
<td>0.22</td>
<td>0.00</td>
<td>2117717/2214013</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.03</td>
<td>0.15</td>
<td>727771/13380947</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.06</td>
<td>0.04</td>
<td>727771/727771</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[65]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td>0.00</td>
<td>1455543/1455545</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.pthread_mutex_lock [78]</td>
<td></td>
<td></td>
<td>0.01</td>
<td>0.00</td>
<td>1455543/1455545</td>
<td></td>
</tr>
<tr>
<td>0.01</td>
<td>0.00</td>
<td>1455543/1455545</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.pthread_mutex_unlock [96]</td>
<td></td>
<td></td>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td>pthread_exit</td>
</tr>
</tbody>
</table>
Selected Techniques

- Maximize Expressions
  - Transforming if-max calculation into using the ?: operator

- Use of registers
  - Using registers to carry values to next iterations, eliminating a large number of load operations

- Fusion
  - Helped increase registers reuse

- Better arrays access
for (k = 1; ...
{
    ...
    ...
if ((sc = dmx[i-1][k-1]...
    ...
    ...
    sc4 = dmx[i-1][k-1] ...
    sc5 = (sc2 > sc1) ? sc2:sc1;
    sc6 = (sc4 > sc3) ? sc4 : sc3;
    sc7 = (sc6 > sc5) ? sc6 : sc5;
    ...
    ...

After maximizing expressions
hmmsearch Timings with a Single Query and the NR Database

Timings with a Single Query and the NR Database

POWER3 375 MHz
POWER4 1.3 GHz

Original
Optimized

3.55x
2.40x
I. Single node optimization
II. Port from PVM → MPI
III. BG parallel optimizations

PVM calls

- `pvm_initsend`
- `pvm_pk (pack) instructions or pvm_upk (unpack)`
- `pvm_send` or `pvm_recv`

- Replaced with MPI calls
  - `MPI_Send` for each `pvm_send`
  - `MPI_Recv` for every `pvm_recv`
  - `memcpy` for every `pvm_pk` and `pvm_upk`
  - `MPI_Send` and `MPI_Recv` to send and receive the entire package

- Functions were constructed to pack the HMM data along with other control structures in parallel with the PVM to MPI conversions

---

Normalized search time vs. number of processors for plain MPI port.
Blue Gene Optimizations

- Alternate Sequence File Indexing
  - Open file and skip to offset
- Multiple-Master Configuration
  - Single master not enough to handle communication
  - Use current infrastructure and include another management level
  - Multiple-master structure is able to do an intermediate processing step
- Dynamic Data Collection
  - Eliminate gather operation
  - Introduce buffer and tolerance (threshold)
- Database Caching in hmmpfam
  - Eliminate excessive I/O
- Load balancing
  - Index file and offset

Parallel performance using the first 327 entries of the Pfam database


HMMSearch parallel optimizations

- hmmsearch parallel performance using 50 proteins of the globin family
- For each processor count, the left bar shows the original PVM to MPI port
- The second bar shows the multiple master implementation.
- The third bar shows the dynamic data collection implementation.
- The right bar shows the load balancing implementation.
Opposite of hmmsearch, but similar in program structure (same optimizations)

In addition to the other optimizations, data caching allowed fast processing of thousands of query sequences

Also scales close to linearly up to 1000+ nodes

- HMM profile: globins
- UniProt Database: ~2.9 million sequences
- Jobs were submitted using LoadLeveler to Blue Gene/P
- 20-25% performance improvement
BLAST®: Basic Local Alignment Search Tool

- A set of similarity-search programs for searching available sequence databases (regardless of whether the query is protein or DNA)
- The most popular tool in bioinformatics
  - NCBI BLAST server: ~ 500,000 query submissions per day
Hexokinase, from the yeast species Saccharomyces cerevisiae

The figure illustrates the growth of GenBank from 1982 to 2005, showing how the database size is increasing faster than our ability to compute on it. The SwissProt database size is also depicted, highlighting the challenge in managing the expanding data volume.
<table>
<thead>
<tr>
<th>Subject(s)</th>
<th>One Sequence</th>
<th>One Sequence</th>
<th>N Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target(s)</td>
<td>One Sequence</td>
<td>M Sequences (in database)</td>
<td>M Sequences (in database)</td>
</tr>
<tr>
<td>Parallelism</td>
<td>Multiple Alignments on Single Sequence Pairs</td>
<td>Partition Database [Multiple targets searched at once]</td>
<td>Replicate Database [Partition Input Sets]</td>
</tr>
</tbody>
</table>
BLAST to mpiBLAST-PIO Evolution

mpiBLAST
- DB is partitioned and BLAST is executed in parallel

pioBLAST
- Uses parallel I/O to improve mpiBLAST
- Dynamic (virtual) DB partitioning
- Improved result merging

mpiBLAST-pio
- Incorporates the parallel-I/O performance enhancements of pioBLAST into mpiBLAST


Importance of mpiBLAST-PIO

Completion of the sequencing of human genome

New organisms being sequenced at a rapid rate
  - NCBI BLAST server: ~ 500,000 query submissions per day
  - Queries per day doubling approximately every year

Trend in GenBank Database
  - Doubling in size every year.
  - Consequence: Database size increasing faster than our ability to compute on it.

What to Do?
  - Faster & more scalable parallel algorithms, i.e., mpiBLAST-PIO.
  - More efficient use of state-of-the-art hardware, i.e., BG/L and BG/P.

Initial Problems

- Disk I/O overload
- Master overworked

---

- Original BLAST version utilizes mmap to store the database in memory
- mmap is not implemented as a part of the Blue Gene/L operating system
- With all nodes sharing the same file system, I/O contention severely limits the scaling of this application
- **Solution**: “virtual file manager“ (VFM)

VFM is used to store:
- database fragments in memory
- query files in memory
- various temporary files in memory

Eliminates disk I/O

Allows files distribution using MPI when workers need the same file

- Second level of management
  - Limit the number of workers for a single master
- Groups of nodes
  - One master for each group working on separate query sequences

Query fragmentation
  - Load-balancing

Multiple output

- Query (28k)
- DB (2.4M)
- Super-master
- Master
- Worker

o **Approach**
  - Exploit the distributed processing power and memory of supercomputing systems, particularly for large datasets.

o **Software Environment**
  - Operating System: Linux
  - Programming Language: C++ and MPI (Message Passing Interface)

o **Overview of Parallel Algorithm**
  - Segment query file into individual queries (only one query shown below)
  - Fragment database and distribute to the worker nodes.
Performance Scaling

- Thick Line
  - Ideal Speed-Up
- Thin Solid Line
  - Speed-up for a large query against nr
- Dashed Line
  - Speed-up for a medium query against nr
- Dotted Line
  - Speed-up for small query against nr

1.5 Performance Improvements

- DB frags cached in workers, queries streamed across
- One output file per partition
- Results merged and written to GPFS through I/O nodes

**Diagram:**

- Partition 1
- Partition 2
- Partition i

- Compute Nodes
- Compute Nodes
- Compute Nodes

- IO Node
- IO Node
- IO Node

- File 1
- File 2
- File i

- Disk
- Disk
- Disk

**Config example:**
- PSize 128
- 4 DBs / partition
- $32768/128 = 256$ partitions
Experimental setup
- Database: NT (over 6 million seqs, 23 GB raw size)
- Query: 512 sequences randomly sampled from the database
- Metric: Overall execution time

WM outperforms WC and WI by a factor of 2.7 and 4.9
Self comparison of Microbial Gnome database (5.2 GB raw size, 16 million sequences)

- Scalability tests
  - Search a quarter million of randomly sampled sequences against the database itself
  - Achieve 93% parallel efficiency on 32768 cores (8-rack BG/P)

- Complete genome-to-genome comparison
  - Finish searching 16 million vs. 16 million sequences within 12 hours

PBPI is an open source implementation of Parallel Bayesian Phylogenetic Inference.
Combines sequential optimization and parallel processing to reduce execution times.
Supports large problem sizes.
PBPI uses MPI (message passing interface) and runs under Linux. Its parallel algorithm can be summarized as:

1. Multi-dimensional data and task distribution across multi-dimensional grid topology organization of processors;
2. Context aware synchronization across the whole grid and sub grid.

PBPI significantly reduces phylogenetic inference time by exploiting distributed processing power and memory, especially for large data set. For proper sizes of phylogenetic problem, **PBPI is capable to scale up to thousands of processors on Blue Gene**.
Scalability on BG/L


• Ligand-receptor docking

Goal:
- Given a protein and a ligand, determine the pose(s) and conformation(s) minimizing the total energy of the protein-ligand complex

Challenges:

- Predicting energetics of protein-ligand binding
- Searching space of possible poses & conformations

- **DOCK 5**: D Moustakas, S C H Pegg, and I D Kuntz in *Virtual Screening in Drug Discovery*, Edited by J. Alvarez and B. Shoichet, Taylor&Francis, Inc.

- **DOCK 6**: P T Lang PT et al. (in preparation)

- **MPP DOCK**: A Peters, M E. Lundberg, C P Sosa, and P. Therese Lang: *High Throughput Computing Validation for Drug Discovery Using the DOCK Program on a Massively Parallel System*, REDP-4410-00 Redpapers, published 16 April 2008
Embarrassingly Parallel

For each independent node, load:
  - DOCK binary
  - Receptor input files
  - Subset of potential drug candidates

Store docking score results into database

Create database of potential drug candidates
Andrew P Norgan¹, Paul S Coffman², Jean-Pierre Kocher¹, David J. Katzmann¹, Carlos P. Sosa²,³

¹Mayo Clinic, ²IBM Corporation, Rochester, MN, ³Biomedical Informatics and Computational Biology, UMR

Andrew P Norgan¹, Paul S Coffman, Jean, Pierre Kocher, David J. Katzmann, Carlos P. Sosa, BICB Research Symposium, University of Minnesota Rochester, June 25, 2010, Rochester, MN
Outline

Biomedical Informatics & Computational Biology

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Summary

Biomedical Informatics & Computational Biology (BICB)
Multiple applications in the area of Life Sciences have been enabled on a massively parallel system.

Sequence alignment Bioinformatics applications can be mapped onto a massively parallel architecture and take advantage of its architectural features.

Multiple optimization techniques were required to improve performance on a single node.

Multiple optimization techniques were required for extreme scalability:
- Alternate Sequence File Indexing
- Multiple-Master Configuration
- Dynamic Data Collection
- Database Caching
- Load balancing

Extreme scalability enables us to complete a large-scale bioinformatics problem — sequence searching a microbial genome database against itself to support the discovery of missing genes in genomes — in only a few hours on BG/P. Previously, this problem was viewed as computationally intractable in practice.


Outline

Part I: Hardware
- Historical perspective: Why do we need MPPs?
- Overview of massively parallel processing (MPP)
- Architecture

Part II: Software
- Overview
- Compilers
- MPI
- Building and Running Examples on Blue Gene
  - Hands-on session 1

Part III: Applications
- MPP architecture and its impact on applications
- Performance tools
- Introduction to code optimization
  - Hands-on session 2
- Mapping applications on a massively parallel architecture
- Applications landscape
- Challenges and characteristics of Life Sciences applications
- Selected Bioinformatics applications
- Selected Structural Biology applications
  - Hands-on session 3

Summary

Biomedical Informatics & Computational Biology (BICB)
Building Partnerships - BICB Biomedical Informatics and Computational Biology

University of Minnesota
ROCHESTER

The Hormel Institute
University of Minnesota

IBM

Mayo Clinic
BICB Objectives

- Establish world-class academic and research programs in bioinformatics and computational biology at UM Rochester.

- Leverage the University of Minnesota’s academic and research capabilities in partnership with IBM, Mayo Clinic, Hormel Institute and other industry leaders.

- Build academic and research programs that complement southeast Minnesota’s existing leadership roles in health sciences, biosciences, engineering and technology.

- Create academic and research programs that provide applications to economic activities via innovation, translational research, and clinical experiences.
Overview: Biomedical Informatics and Computational Biology (BICB)

- Interdisciplinary, all-University graduate program
  - University of Minnesota Twin Cities
  - University of Minnesota Rochester (administrative home)
- Ph.D. and Master of Science (M.S. Plan A and Plan B) degrees and a Minor
- Graduate faculty are from
  - University of Minnesota Twin Cities
  - University of Minnesota Rochester
  - Hormel Institute
  - Mayo Clinic
  - IBM
- Students are in residence on either the Rochester or Twin Cities campus
- The program is suitable for full-time and part-time students
Admission Requirements

- Strong background in the quantitative sciences and varied backgrounds in the life/health sciences
  - Calculus (1 year)
  - Introduction to computer science or programming (1 semester)
  - Chemistry (1 year)
  - General Biology (1 semester)

- Background in either two of the areas 1-3 or one of the areas 1-3 and one of the areas 4-5
  1. Multivariable calculus, differential equations, linear algebra
  2. Algorithms & data structure, discrete mathematics
  3. Statistics or biostatistics, probability theory
  4. Biochemistry, genetics, and cell biology
  5. Health sciences (pharmacology, physiology, or related areas)

- Deficiencies must be made up during the first year
BICB Graduate Program

CORE AREAS:
1. Biochemistry, molecular and cell biology
2. Database, data mining, and computing
3. Informatics, analysis, and machine learning
4. Mathematics, biostatistics and statistics
5. Computational and systems biology

ELECTIVE AREAS:
1. Biochemistry, molecular and cell biology
2. Informatics, database, data mining, and computing
3. Mathematics, biostatistics and statistics
4. Chemistry, chemical engineering, and physics
5. Biophysics and structural biology
6. Imaging, information theory, and signal processing
7. Computational chemistry, medicinal chemistry and drug design
8. Clinical and translational science
BICB Graduate Program

• Personalized degree program to meet the needs of full-time and part-time students
  • M.S. Degree
    • Course-work plus capstone or course-work plus thesis
  • Ph.D. Degree
    • Interdisciplinary and collaborative research environment
    • Internships
    • Professional development (leadership and management skills)
    • Mentoring

CONTACT INFORMATION:
Professor Claudia Neuhauser - Director of Graduate Studies
Vice Chancellor for Academic Affairs, UMR
Telephone: 507-281-7791
E-mail: neuha001@umn.edu
The Breadth of Research in BICB

- Data mining of biomedical data
- Metabolic pathways
- Mining of unstructured biomedical data
- Screening for drug development
RNA Catalysis
Prof York’s Group (U of MN)

Metabolic Pathways
Prof Boley’s Group (U of MN)

Ebola Virus Therapeutics
Prof. Kaznessis (U of MN)
Dr Kocher’s Group (Mayo Clinic)

Life Sciences Environment for Blue Gene

Kinases Small Molecules Inhibitors
Dr Dong & Dr Bode’s Group (Hormel Institute)

Dimitrije Jevremovic, Ph.D. Candidate, Computer Science

Emilia Wu, Post-Doc, Chemical Engineering

Rashed Ferdous, Ph.D. Candidate, IBM

Andrew Norgan, Ph.D. Candidate, Mayo Clinic

Madhusanan Mottamal, Post-Doc, Hormel Institute

Geroge Giambasu, Ph.D. Candidate, Chemistry
BICB Resource

IBM JS22

IBM Blue Gene/P

MSI

IBM Blue Gene Center Rochester
**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 Location

Silver Calhoun Blade Itasca SDML SDVL BSCL
Workshops

- **Introductory**
  - Unix, Linux, remote computing, job submission, queue policy

- **Programming & Scientific Computation**
  - Code parallelization, programming languages, math libraries

- **Computational Physics**
  - Fluid dynamics, space physics, structural mechanics, material science

- **Computational Chemistry**
  - Quantum chemistry, classical molecular modeling, drug design, cheminformatics

- **Computational Biology**
  - Structural biology, computational genomics, proteomics, bioinformatics
- The institute’s web page
  - [https://www.msi.umn.edu](https://www.msi.umn.edu)

- Getting started
  - [https://www.msi.umn.edu/support/start.html](https://www.msi.umn.edu/support/start.html)

- Software
  - [https://www.msi.umn.edu/sw](https://www.msi.umn.edu/sw)

- Password reset
  - [https://www.msi.umn.edu/password](https://www.msi.umn.edu/password)

- Tutorials
  - [https://www.msi.umn.edu/tutorial](https://www.msi.umn.edu/tutorial)
BICB: Grants and Traineeships
Hormel Institute: Blue Gene
Mayo Clinic
IBM: Blue Gene Rochester and Watson
University of Minnesota Supercomputing Institute (MSI)
Thank You!