

Exploring Computational Biology with a Massively Parallel High Performance Computing Environment

Kirk E. Jordan

IBM

1 Rogers Street

Cambridge., MA 02142

Srinivas Aluru

Department of Computer Sciences

Iowa State University

Outline

In this tutorial, we present the audience with evidence that massively parallel computing environments will and are playing a significant role in scientific discovery in the biological sciences. The infrastructure and the knowledge to take advantage of this infrastructure is presented along with pointers for added information and help to employee massively parallel computing infrastructure to accelerate bioinformatics and computational biology research

Need to explain this

Computation is playing an ever increasing and vital role in biology creating demand for new machines. Vendors strive to meet demands with advanced computer architectures such as IBM's Blue Gene machine. In this tutorial, we will give an overview of the Blue Gene architecture. We will briefly describe both the hardware and software architecture and the central philosophy behind the development of the Blue Gene that makes it easy to use on ultrascale problems. We will emphasize the key features that allow thousands of processors to work together on a user's problem. We will present the programming model used on Blue Gene. We will explain ways to take advantage of the Blue Gene nodes and their associated networks. We hope to provide a foundation for attendees to begin to think about problems and how to design and implement them so they will scale out and take full advantage of the computational power in Blue Gene.

Once we have presented a basic understand of the architecture, our goal will be to show how Blue Gene is impacting bioinformatics through several examples. We will describe briefly some solutions done on Blue Gene in such as areas as protein folding, transcription factor binding sites, and systems biology to demonstrate to the audience the wide applicability. We will try to illustrate the ease of use of the systems through remote demonstration if Internet facilities are available. Depending on the partition size, we will demonstrate some simple scaling up to the number of processors available on some simple problems pertinent to the audience.

Through the computational power of Blue Gene, scientists will tackle problems that to date they had not considered. This will happen in some ways we are starting to see today but there are other approaches we yet can not predict. For this reason, we will discuss alternative approaches to design of computation of the problem that might spark others imagination. We will also show by example some problems that

one might not think would be suitable for the Blue Gene architecture. We will discuss in these examples which have been run on Blue Gene systems, actual performance results. More importantly, we will try to point how having unprecedented number of processors changes how one approaches the computational problem.

The tutorial will proceed to go in depth on one application area, Genome Assembly. We will describe a massively parallel framework for genome assemblies on the Blue Gene, and its application to the ongoing maize genome sequencing project. We will show how to harness the power of the massively parallel system, Blue Gene, to carry out genome assemblies at a significantly rapid pace of hours instead of days and weeks. We will discuss the applicability of this framework to solve other large-scale computational genomics problems including EST clustering, SNP identification, and selected problems in comparative genomics.

If successful, the audience will leave this tutorial with sufficient knowledge of how massively parallel or ultrascale out computing can accelerate their research in the biological sciences. While some, based on their own experience and computational background, may be able to put this knowledge to immediate use taking advantage of opportunities mentioned for access, others will have sufficient information to know where to get more details and possibly even to form new collaboration to accelerate their research. The intention of this tutorial is to use examples of bioinformatics and computational biology to demonstrate the utility of ultrascale out computing while dismissing the myths associated with massively parallel/ultrascale out computing.

CONTENTS

INTRODUCTION.....	6
Background	6
Massively Parallel Computing Making a Difference	8
 COMPONENT ARCHITECTURE	 10
Massively Parallel (Ultrascale Out) Computing Environment.....	10
Blue Gene/L Building Blocks	10
The Blue Gene/L System	11
The Blue Gene/L Networks	12
The Blue Gene/L Processors	14
 SOFTWARE AND PROGRAMMING MODEL OVERVIEW	 15
System Software	16
The Blue Gene/L System Overview	17
Programming Models and Development Environment.....	18
The Blue Gene/L Processor Execution Modes	19
 PERFORMANCE OPTMIZATION AND TOOLS	 20
Performance Optimization and Libraries	20
Performance Decision Tree	21
Compiler Options and Switches	21
HPC Toolkit Overview	21
Message Passing Performance	22
Processor Performance	22
Hardware Performance Monitors – Instrumented Code.....	22
Running Code on Blue Gene/L	23
 APPLICATION PERFORMANCE.....	 25
Blue Matter Framework	25
New Science and Outstanding Performance	26
 MASSIVELY PARALLEL COMPUTING ENVIRONMENT.....	 27
Massively Parallel Computing Environment - What Next?	27

LARGE-SCALE COMPUTATIONAL GENOMICS ON THE IBM BLUE GENE/L SUPERCOMPUTER	28
Genome Assembly	28
Expressed Sequence Tag	29
Single Nucleotide Polymorphisms - SNP	29
Naïve Approach to Alignment	30
Lookup Tables	30
PaCE Methodology	30
MAIZE GENOME ASSEMBLY	31
Maize Genome	31
Assembly Strategy	31
Maize Genome Assembly Effort	31
Cluster-Then-Assemble	32
Pair Generation	33
PaCE Methodology	34
Pair Generation Algorithm	34
Master-Worker Paradigm	35
Assembly Pipeline	35
Blue Gene Performance	35
Maize/Sorghum Assembled Genomic Islands	35
Maize Genome Project Future	36
MOUSE EST CLUSTERING.....	36
Mouse EST	36
Validating Accuracy	37
CLOSING REMARKS.....	37
Blue Gene Consortium	37
Final Remarks	38
<i>REFERENCE LIST/ADDITIONAL READING</i>	<i>39</i>
<i>Additional WEB Sites</i>	<i>43</i>

INTRODUCTION

Background [Slides 3-8]

Over the course of the years in computing, we have seen several trends develop in high performance computing. Some of these trends have been the result of the chip technology while other trends have been the result of innovative assembly of processor technology. As technology hits a wall, innovation allows us to take new approach. Massively parallel computing is not new, but innovation is allowing for truly massive parallel computing in dense packaging while keep power consumption and cooling requirements at a minimum.

The number of transistors on a computer chip has doubled every couple of years. This is called Moore's law. What this meant is the number of floating point operations per second (flops) a computer could perform has also increased. Eventually the constraint on the overall size of a single computer chip and the physical limitations on how small a transistor could be produced have to stop that curve.

Shrinking transistors has an absolute limit, which we are approaching, and also yield increasingly difficult side effects such as power leakage. In order to continue to get increased performance, we turn to the clustering of chips together to allow the continued increase in the number of flops. This led to the development of computers with numerous CPUs sharing the same memory requiring some very fast and sophisticated interconnects that increase the system cost as the number of CPUs within these shared memory machines increases.

With commodity computing in the 1990's, the cost of large scale machines giving increased flops could be achieved using individual CPUs networked, or clustered, to function together as a single unit. This class of systems became known as massively parallel processing (MPP) systems. The only theoretical limit to their size was the floor space, power consumption, and cooling needed to house and run the aggregated equipment.

From the application point of view it became apparent that the limitation on increased flops depended not only on the individual performance of the CPUs but also on the performance of the entire system on which the CPUs depend including

memory system, the file access and network (messaging). It also became clear that these types of systems could not handle every application. As the number of processors increases taking advantage of them becomes more difficult, and there are some type of applications that cannot take advantage of the extra power. But for those that do, developers and users need access now to larger numbers of CPUs to find ways to scale their applications to ever higher number of processors.

A massively parallel (MPP) system in general has the following characteristics:

- A single system image for up to thousands of nodes.
- The cost per flop is extremely low because each node is an inexpensive processor.
- Each node has its own distinct uniquely addressable memory.
- The nodes are organized into a grid, mesh, torus or hypercube arrangement to allow each node to communicate with all the other nodes.

The aggregate MPP system has access to a huge amount of real memory for the application operations to access, because this is the sum of the memory available to each node.

The parallelism we see in systems used for technical computing is pervasive. We see this commercial off-the-shelf (COTS) systems assemble into large clusters. Hybrid systems, combining commercial processors with high speed interconnects, are available. Each of these kinds of systems is limited on the ability to obtain ultrascale out. Custom architectures such as the IBM Blue Gene system are addressing the environmental requirements while providing ultrascale out systems.

The IBM eServer Blue Gene Solution is not a completely new system with exotic components. On the contrary, the processors of the Blue Gene/L system come from IBM's family of embedded PowerPC processors. They are enhanced for the kind of work loads with heavy floating point computation expected in high performance computing. The use of embedded processor technology keeps the computing environmental requirements low.

In addition, the Blue Gene/L is not a radically different architecture for IBM. Looking at computer systems in a two dimensional space consisting of scale up systems versus scale out systems, we see that Blue Gene/L is at the high end of the scale out systems. For this reason, we often refer to it as an ultrascale out system. Placing Blue Gene/L in the IBM high performance computing portfolio, we see that it

is design for those that need capabilities that can only be satisfied by purpose built systems which allows for ultrascale out.

Massively Parallel Computing Making a Difference [Slides 9-14]

Before delving into a lot of detail of a massively parallel computing environment, it is worthwhile to demonstrate such an environment is of wide value in bioinformatics and computational biology. Over the past year, the first author has enabled and coached members of Charles DeLisi's group at Boston University. This has resulted in the implementation of the Gigtigs code on a massively parallel computing environment, the IBM eServer Blue Gene Solution.

The human body consists of some 200 major cell types – various types of neurons, blood cells, epithelial cells and so fourth – all of which differentiated from a single blueprint encoded in the genome of a fertilized egg. The process of differentiation itself, which leads to specialized cell function, remains poorly understood, but it is generally believed to involve chromosomal remodelling induced by very tight protein binding and structural modification that blocks the expression of certain sets of genes, while allowing expression of others. Understanding how genes are selected – that is, cracking the genomic regulatory code – is a primary knowledge gap between a genome sequence and the diversity of life encoded therein.

A computational approach has been taken to understand genomic regulation – the code dictating the regions of the genome transcribed into molecular messengers. The transcribed messengers become the proteins that define and control the cell. Proteins regulate the initiation of gene expression by binding short nucleotide sequences generally found upstream of the gene. A typical eukaryotic gene has on average sites for 6-9 different regulators, and each regulator can bind upstream of multiple genes. The result is a complex regulatory network. A first step and more tractable problem is to find the binding sites, about 200 as compared with 600 for mammalian cells, for the simplest eukaryotes, yeast in particular.

Computational attempts to detect the DNA sequence patterns recognized by regulators is to look for short sequences of DNA that exist far more frequently than would be expected at random. Unfortunately, the problem is difficult because the fragments are found in a relatively large amount of unrelated and noisy DNA. At the same time, the DNA sequences of the fragments to which a particular regulator

binds are not always identical. As such, even the best computational approaches have met with considerable difficulty, especially in complicated organisms such as humans.

Tim Reddy and Boris Shakhnovich, from the Biomolecular Systems Laboratory (DeLisi) at Boston University have developed a code for cracking that first key step in genomic regulation. The code, GibTigs, uses Gibbs Sampling, in a novel and exhaustive manner. Current Gibbs sampling implementations generally attempt to crack the code a few times and return the best answer from those attempts. This results often in missing subtle, solutions while producing incorrect answers without biological significance. GibTigs attempts to crack the code several thousands of times, retaining all solutions, and identify potential sites by assembling overlapping solutions. The result is a clear set of signals in a background of noise. By increasing the number of attempts, GibTigs is able to distinguish between incorrect sequences, which tend to appear randomly within the DNA, and correct results, which tend to occur in a small, conserved set of positions every time they are found. From early successes, the group has focused on much larger problems, such as identification of key regulatory sequences across entire genomes as well as exploring evolutionary models of the sequences.

While an individual Gibbs Sampling analysis requires relatively little compute time, iterating the process exhaustively and on a large scale is computational challenge. The feature of GibTigs that allows easy and productive deployment on cluster-based supercomputers is that the computational core of the algorithm depends on thousands of independent iterations of Gibbs Sampling, a directed statistical sampling of the DNA sequence data, requiring much CPU, but little memory. Each iteration can be performed independently allowing for the distribution of tasks over a large cluster of CPUs with minimal overhead from inter-processor communication. This Makes GibTigs ideal for the Blue Gene/L system.

One Blue Gene/L rack represents nearly a twenty fold increase in available compute power, over a conventional cluster available at Boston University. With GibTigs showing linear scalability up through 2048 CPUs, one Blue Gene/L rack (in virtual-node mode) sped development from a few runs a week to many runs a day, enabling large scale parameter searches, and regular production grade performance evaluations. The results are dramatic improvements in sensitivity and specificity of the algorithm, none of which would have been possible at the previous development

pace. Moreover, rather than making conservative modifications to GibTigs, the power of Blue Gene has given the team the freedom to take risks in trying new ideas, many of which have failed, but some of which have provided new insight and new power to GibTigs. As a result, GibTigs has recently proved to be, according to published measures, the most powerful predictor of DNA transcription regulatory sites to date.

COMPONENT ARCHITECTURE

Massively Parallel (Ultrascale Out) Computing Environment [Slides 15-29]

In this section, we describe both the hardware and software philosophy that has led to an ultrascale out machine. With the preceding example and the detailed examples that follow, we believe that this ultrascale out environment will enable the solution of many important problems in bioinformatics and computational biology. Those that learn how to take advantage of this ultrascale out environment as we are already will lead in the breakthroughs and help to steer the directions of discovery in the biological sciences.

Blue Gene/L Building Blocks [Slide 16]

Based on IBM's Power architecture, the IBM eServer Blue Gene Solution is optimized for bandwidth, scalability and the ability to handle large amounts of data while consuming a fraction of the power and floor space required by today's fastest systems. It is an ultrascale computer.

Blue Gene/L is a massively parallel machine. It is as a collection of small basic elements, Power architecture chips, connected together by a set of networks. Starting with the base elements, we show how these are package to build a systems up to 64 racks, to become the current fastest computer.

The Chip: The base component of Blue Gene is a dual core Power PC CPU chip. The CPU frequency is 700 MHz and each CPU can perform four floating point operations per cycle. The theoretical peak performance of each CPU is 2.8 Gflops. The 2 CPUs on the chip constitutes the compute node with a peak performance of 5.6 Gflops.

The Compute Card: The compute nodes are soldered to a processor card. Each processor card has two compute nodes on it. The memory for each chip sits on the other side of the processor card; there are either 512 MB or 1 GB per node (1 GB or 2 GB RAM per compute card) depending on the system. The original Blue Gene System node cards only had 512 MB per node.

The Node Card: The processor cards are plugged on a node card consisting of two rows of eight compute cards. On the node card there maybe two or four I/O nodes but some node cards may not contain any I/O nodes. The I/O nodes are similar to compute nodes; they also sit in pairs on a small processor card which is slightly different from the one used for compute nodes. They are called I/O nodes because their role is different from the compute node role; it is solely for handling I/O.

The Midplane: The processor cards, which bear 16 compute cards, are stacked in a midplane which sits in a rack.

The Rack: A rack holds two midplanes, for a total of 32 compute cards.

The System: One can connect up to 64 racks for a Blue Gene/L system.

To calculate the number of processor in a system, the following formula is used:

The number of racks x number of node cards per rack x number of compute cards per node card x number of processors per compute card,

or

the number of racks x 32 x 16 x 4,

or

the number of racks x 2048.

The largest possible configuration is $64 \times 2048 = 131072$ processors.

Even though the I/O nodes are made of the same PowerPC chip as the compute nodes because they do not contribute to the actual computation, they are not factored into the processor count.

The Blue Gene/L System [Slides 17-18]

The description above gives details of what goes into a Blue Gene Rack. Essentially, the rack contains the compute part of the system. To make a complete system, there are some other components. There are often two additional computers that with the rack make up the Blue Gene System: a service node and front-end node. In addition, file servers with appropriate storage are part of the system. A typical configuration is given in order to understand how all the components come together to really make up the massively parallel computational environment that will be used.

The Blue Gene/L Networks [Slides 19]

As described above, the Blue Gene/L machine is a massively parallel computer. In order for the CPUs to work on data together they must be able to communicate with one another. To accomplish this task, Blue Gene/L needs a communication fabric or network. Actually, Blue Gene/L has 5 networks. The 5 networks can be grouped in two categories, those for efficient parallel coding and those to communicate with the outside world. For parallel efficiency there are two characteristics to keep in mind, bandwidth and latency of the network. The efficient parallel coding networks of most important to the application programmer are the torus network, the global tree network and the global barrier network. For communication to the outside world, there is the gigabit Ethernet network and the control network. Each of these is described below.

The 3D Torus Network: On Blue Gene/L instead of using a cross bar switch for point-to-point communications, a 3D (dimensional) torus network is used. Each node of the torus communicates with its six nearest neighbors through a bidirectional network. In this manner, the 3D torus forms the communications backbone for computations and connects all compute nodes (65,536 on a 64 rack system). The characteristics of the 3D torus are:

- Virtual cut-through hardware routing.
- On each of the 12 connections of a compute node, 1.4Gb/s or 2.1 GB/s per compute node.
- The hardware Latency (Nearest Neighbor) is 200nanoseconds (32B packet), 1.6 microseconds (256B packet). The worst case for 64 hops is 6.4 microseconds.
- The bisection bandwidth is 0.7/1.4 TB/s and a total bandwidth of 67 TB/s.

The Global Tree (Collective) Network: Blue Gene/L has a special network devoted to MPI collective operations such as all-to-all, all-to-one, and one-to-all. This is the Global Tree or Collective Network. This network connects all the compute nodes in a shape of a tree. The root can be any node. The IBM implementation of MPI on Blue Gene/L will use the tree network whenever it is more efficient than the 3D torus for a collective communication. The application developer when using the MPI collective operations will get the most efficient path. The characteristics of the tree network are:

- Has one-to-all broadcast and reduction operations functionality.
- The bandwidth of each link is 2.8 Gb/s.
- The latency of a tree transversal 2.5 microseconds and approximately 23TB/s total binary tree bandwidth on a 64K compute node machine (64 racks).
- The binary tree interconnects all compute nodes and I/O nodes.

The Barrier (Global Interrupt) Network: On a very large system, there is often a need to synchronize or bring every processor to the same point before moving on. Since such communications require small amount of bandwidth but need very low latency, a special network, the Barrier or Global Interrupt network, on Blue Gene/L is provided to handle MPI synchronization routines like barriers or waits. The characteristics of this network are:

- The latency is 1.3 microseconds for a round trip on the network.

All interactions between the outside world and Blue Gene/L go through the service node to the I/O nodes. There are two networks connecting the service node to the I/O nodes. The two networks that connect the service node to the I/O nodes are the Ethernet network and the Control or JTAG network.

The Ethernet Network: This network is used to mount the global file system to allow Blue Gene/L access to I/O. The link from I/O node to compute nodes is the tree (collective) network. The global file system only has to be “global” to all the nodes in a partition, plus the service node and the front-end system used to submit the job.

The Control (JTAG) Network: The control network is used to give the service node direct access to the Blue Gene/L compute nodes. It is used to boot the nodes.

The connection between the node cards and the service node is a 100 Mb/s Ethernet network.

The Blue Gene/L Processors [Slides 20-21]

The compute nodes are comprised of 2 dual core chips. Each core or ASIC is a complete System-On-a-Chip. The Blue Gene/L compute ASIC chip includes two non cache-coherent microprocessors, each containing one single load/store unit, one single 32-bit integer unit and one double Single-Instruction-Multiple-Data (SIMD) 64-bit FPU. Each FPU can execute up to two multiply-adds per cycle, meaning that the peak performance is eight 64-bit floating-point operations per cycle. That is 2.8 Gflops/s per core and 5.6 Gflops/s per chip.

The ASIC block diagram shows the details of two processors, each having a special double floating point unit, connecting individually to L2 cache, and accessing the L3 controller to connect to the L3 cache. The integrated networks on the ASIC, as described before, include:

- six 1.4 Gbit/s bidirectional ports for 3-dimensional torus network connection
- three 2.8 Gbit/s bidirectional ports to a tree (collective) network connection
- one gigabit network connection
- one Joint Technical Advisory Group (JTAG) control and monitoring network connection
- one barrier (global interrupt) network connection

The basic elements of the PowerPC 440 microprocessors are:

- 32-bit architecture at 700 MHz.
- Single integer unit
- Single load/store unit
- L1 cache: 32KB total size, 32-Byte line size, 64-way associative, round-robin replacement
- L2 cache: prefetch buffer, holds 16 128-Bytes lines
- L3 cache: 4MB, approximately 35 cycle latency, on-chip
- Special double floating pointing unit
 - 32 primary floating point registers, 32 secondary floating point registers that support –
 - standard PowerPC instructions which execute on primary registers such as fadd, fmadd, fdiv, ... and
 - special SIMD instructions for 64-bit floating point numbers which execute on the primary and secondary registers such as fpadd, fpmadd,
 - The floating point pipeline is 5 cycles
 - The floating point load-to-use latency is 4 cycles

The Blue Gene/L Double Floating Point Unit [Slide 22-24]

The Double Floating Point unit of the Blue Gene/L Processor has two pipes. The primary pipe executes the standard instructions and the SIMD (Single Instruction Multiple Data) instructions while the second pipe only executes the SIMD instructions. The double FPU (Floating Point Unit) implemented on Blue Gene/L chip offers more capabilities than a pure SIMD unit. Some instructions cause two different operations to be performed in the two pipes. For example, the instructions allow to efficiently support complex cross products. Other instructions cause a single operation to occur on a single set of data.

The results from the pipes are only written to the corresponding FPRs (Floating Point Registers), primary FPRs for the primary pipe and secondary FPRs for the secondary pipe. However, the cross micro architecture of FPU allows the primary and secondary pipes to select primary FPR values or secondary values. Each pipe has 5 stages and can execute one multiply-add per cycle.

Although, there are two sets of register files, they are not independent and share address buses for each port. The secondary FPR is accessed with the same addresses as the primary FPR. The optimal way to fill out the FPRs is to access the operands in pairs, one primary and one secondary. The Load/Store pipe of the double FPU makes full use of the quadword APU interface. One load and store can provide two double-precision operands or two single-precision operands, one for the primary and one for the secondary pipe. To achieve this, the memory accesses must be quadword aligned.

In order to understand the impact of the double floating point units as a brief aside, we show the impact of the performance on a DAXPY, double precision scalar times a vector plus another vector, kernel. This is a fundamental kernel often found in the Basic Linear Algebra Subroutines, BLAS, which are optimized for a particular architecture and called from a library.

In this example, give the source code in both C and Fortran. The graph illustrates the impact of both aligning the data to be quad word aligned and use of the secondary floating point unit when aligned. Note that compiler switches are used as well as directives in the code.

SOFTWARE AND PROGRAMMING MODEL OVERVIEW

The fundamental philosophy behind the systems software is simplicity in order to scale to tens and hundreds of thousands of processors. In addition, a familiar application interface that will allow users to easily move existing codes to the system. With this in mind, the systems software developers avoid features in the operating systems that were not essential for high performance computing and strove for simplicity to achieve both efficiency and reliability.

The system software consists of:

The Compute Node Kernel: The kernel that runs on the compute node is called Compute Node Kernel (CNK). This is a small simple kernel that provides a Linux-like simple runtime environment to run the user's application. It is IBM proprietary. It does include a subset of Linux system calls primarily to handle I/O so the end-user can open and close, read and write, create directories, etc. This kernel is Single user, Single process and no paging. The Compute Node, as mentioned, communicates to the outside world through the I/O Node. The executable program is loaded from the I/O node through the Collective network.

The I/O Node Kernel: The kernel of the I/O node is called Mini-Control Program (MCP). It is a port of Linux Kernel which means it is GPL/LGPL licensed. It has specific patches for the Blue Gene Architecture such as:

- Patches for Blue Gene/L
- New interrupt controller (BIC)
- Save and restore for dual FPU registers on context switch
- New memory layout
- New set of Device Control Registers (Dicers)
- Driver for new Ethernet Macro (EMAC4 based on EMAC3)

The I/O service is provided through the Compute Node I/O Daemon (CIOD) on the Compute Nodes. It is started during the boot procedure of the MCP. The CIOD is a user level process which controls and services applications in the Compute Node and interacts with the Midplane Management and Control System (MMCS).

The Midplane Management and Control System(MMCS): Both Blue Gene/L hardware and software are controlled and managed by the Midplane Management and Control System(MMCS). The service node, front-end nodes and the file servers

are not under the control of MMCS. MMCS currently consists of several daemons which interact with a DB2 database running on the service node.

The Service Node Blue Gene/L Software: The Service Node has three daemons running on it that perform Blue Gene system management services. These daemons are **idoproxydb**, **mmcs_db_server** and **ciodb**. These three daemons perform the following functions:

- **idoproxydb** - handles the communication to the cluster hardware.
- **mmcs_db_server** - manages the blocks(also known as partitions), handles the requests from mmcs_db clients (mmcs_db_console, mmcs_db command scripts or a job scheduler).
- **ciodb** - detects the block when it is initialized and manages the job submission request.

There are four DB2 databases on the Service Node that interact with the MMCS. These are:

- **The configuration database** which records Blue Gene/L component location and connectivity. Most items in this database relate to specific physical pieces of hardware.
- **The operational database** which records partitions, job status, and events related to ongoing Blue Gene/L system activity. (Although it called one of the four databases, the operational database is actually part of the configuration database.)
- **The environmental database** which records periodic readings of voltage levels, switch settings, and sensors.
- **The Reliability, Availability, Serviceability (RAS) database** which records both software- and hardware-related errors. It is the RAS database which is most closely watched by system administrators keeping an eye on the overall system health.

The Blue Gene/L System Overview [Slide 27]

Taking all the above into account, we see that the Blue Gene/L system is really comprised of several computers, the Service Node, the Front-End Node(s), File Servers all connected with the Blue Gene rack which contains the compute and I/O nodes. Summarizing these systems perform the following functions:

- **Service Node** - Used for controlling the Blue Gene/L system.
- **Front End Nodes** - Users log in to these nodes and submit jobs to the Blue Gene/L system.
- **Compute Nodes** - The compute engines inside the Blue Gene/L racks.
- **I/O Nodes** - Installed inside the Blue Gene/L racks.

- **File Servers** - Provide a file system accessible both by the Front End Nodes and by the I/O nodes.
- **Functional Network** - A common network used by all components of the Blue Gene/L system except the Compute Nodes.
- **Control Network** - Used for specific system control functions between the Service Node and the I/O nodes.

Programming Models and Development Environment [Slide 28]

One of the goals of the Blue Gene Project was to create a system that was familiar to the application developer. To achieve this, a message passing programming model built on MPI (Message Passing Interface) was adopted. Most application programmers familiar with programming on clusters using MPI would be familiar with the Blue Gene environment.

The development environment should be familiar to many programmers. The programmer interacts with the Blue Gene system through the front-end node. The front-end node runs the Linux operating system. The users does all development, compilations, job submission, and debugging on the front-end node. The compilation for Blue Gene is done through the use of a cross compiler, the user compiles on the front-end or other supported platform but targets the executable through appropriate compiler switch to run on Blue Gene. Supported languages are Fortran, C, and C++ with MPI (MPI1 plus an appropriate subset of MPI2). The compilers also support automatic generation of the SIMD instruction for double floating point unit.

For the most part, the programming model on Blue Gene is a Single Program Multiple Data (SPMD) model. This, too, is familiar to many who program clusters. Numerous applications written using MPI calls fit into this model making it easy to move codes over to try out Blue Gene.

In addition, there are numerous tools familiar to application developers and more tools are being added. Some familiar tools include:

- Debuggers – TotalView
- Profiling and trace tools – MPI Tracer
- Hardware performance monitors – HPC Toolkit, Paraver, Tau, Kojak

You can find more information about these tools and the status of them and others at the following Websites:

- TotalView (Etnus @ Entus LLC)
<http://www.etnus.com/TotalView/>
Parallel debugger
- PARAVR (UPC @ U of Barcelona)
<http://www.cepba.upc.es/paraver/>
KOJAK (ICL @ U of Tennessee and ZAM @ FZ Jülich)
<http://www.fz-juelich.de/zam/kojak/>
Kit for Objective Judgement and Knowledge-based Detection of Performance Bottlenecks
- PAPI (ICL @ U of Tennessee)
<http://icl.cs.utk.edu/papi/>
Performance Application Programming Interface
- TAU (U of Oregon)
<http://www.cs.uoregon.edu/research/paracomp/tau/tautools/>
Tuning and Analysis Utilities
- mpiP (LLNL)
<http://www.llnl.gov/CASC/mpip/>
Lightweight, Scalable MPI Profiling

While the application development should be familiar to many, there are a few things to keep in mind while developing code to run on the Blue Gene/L Compute Nodes. One feature on running a code on Blue Gene is the system is strictly a space sharing system. This means that there is one parallel job (user) for each partition of the machine. Further, there is one process per processor of the compute node.

The virtual memory of the system is constrained to the physical memory. This requires the developer to understand the memory requirements of the application. It is wise to manage memory carefully on this system.

The Blue Gene/L Processor Execution Modes [Slide 29]

The Compute Nodes are composed of a pair of CPUs in a single chip, supporting chips, and 512 MBs (1 GB) of dedicated memory in which the user's application runs. The application user may set up at the time the partition is booted to use the two processors in one of three ways: Coprocessor Node Mode, Virtual Node Mode, or Hybrid Node Mode.

- **Coprocessor Node Mode** is a configuration that uses the secondary CPU as an offload coprocessor for processing the I/O of the main CPU. This reduces the burden on the main CPU, and provides all 512 MB (1 GB) of memory for the user application in that CPU. Coprocessor Node Mode, will not assist with file based I/O, only messaging, and only then after the primary node starts it.
- **Virtual Node Mode** is a configuration that uses both CPUs separately, running a different instance of the user's application on each processor. In this mode, the 512 MB (1 GB) memory is split between the two processors, giving each processor effectively 256 MB (512 MB) of memory for the Compute Node Kernel and user application. Each processor also handles its own I/O interactions for messages and the file system I/O stubs.
- **Hybrid Node Mode** is a non-default configuration created by the coder. It sometimes is referred to as "Communication Coprocessor Mode with Computation Offload". In this mode, the secondary processor functions as both an I/O coprocessor and as user application processor. This mode is of use for those who don't mind coding their own behavior and the details that go with performing such a task, details such as handling the lack of L1 cache coherence between the two processors, in order to wring out the last 2-4% of speed possible in the Blue Gene/L system.

PERFORMANCE OPTIMIZATION AND TOOLS

Performance Optimization and Libraries [Slides 30-32]

To develop programs on Blue Gene/L, one needs to compile their code. The IBM XL compilers for Fortran, C and C++ are available. In addition, IBM's batch processing scheduler called LoadLeveler is available for dispatching jobs. For I/O performance, the General Parallel File System (GPFS) is available for Parallel I/O to external disk. Several math libraries have been optimized for Blue Gene, The Engineering and Scientific Subroutine Library (ESSL) is ported to Blue Gene and most functions are available in highly optimized form. The MASS and MASSV library provide an additional performance benefit via highly optimized intrinsic functions. The comparison between the default libm.a and the MASS and MASSV library for several intrinsic functions is illustrated in the table below.

Cycle count per evaluation on Blue Gene/L processor

Function	Libm.a	Libmass.a	Libmassv.a
Exp	185	64	22
Log	320	80	25
Pow	460	176	29-48

Sqrt	106	46	8-10
Rsqrt	136	...	6-7
1/x	30	...	4-5

The relative costs of operations on the Blue Gene/L processor are given in the table. It shows that division is the costliest primitive operation. Avoiding division, one can achieve 1 mult-add per cycle on each pipe on each of two processors, unless limited by load-stores. There is no hardware square-root function on Blue Gene/L and the default GNU library function from libm.a is slow. The recommendation for codes dominated by SQRT is to use the IBM supplied MASS and MASSV libraries which are available for download from the following URL:

http://www-1.ibm.com/support/docview.wss?rs=2021&context=SSVKBV&dc=D400&uid=swg24009222&loc=en_US&cs=UTF-8&lang=en .

Performance Decision Tree [Slide 33]

On the Blue Gene/L system, there is a variety of tools one uses to obtain and improve code performance. These tools and their connection are outlined the Performance Decision Tree. Total performance comes from computation, communication and I/O performance. The tools available for each are list in this tree diagram.

Compiler Options and Switches [Slide 34]

For computation, one should not forget the compiler. A lot of performance improvement can be obtained through a variety of compiler switches. There are various levels of optimization O through O5. With the wealth of compiler switches and options, it is hard to determine what will work best in all cases. Here we recommend a specific starting point of -g for debugging purpose, -O for minimal optimization purpose, -qarch=440 for the architecture and -qmaxmem=64000. The -qarch is important because we are doing cross-compiling, in other words, we are compiling on another PowerPC architecture and targeting for the 440PC.

HPC Toolkit Overview [Slides 35-36]

IBM has an integrated tool kit, the HPC Toolkit. Couples several tools, a profiler, Xprofile, to understand the performance of code on a processor; an interface to the PowerPC chip monitors, HPM; and a message passing profiler, MP Profiler and Trace facility to understand communication. Work is underway to include profilers for I/O. All these performance monitoring tools are integrated in a common viewing tool called Peekperf which aids in visualizing and analyzing the performance.

Message Passing Performance [Slides 37-39]

In the first set of slides, we show the information on message passing performance. Information can be obtained through the MP_Profiler library and the MP Trace library. The profiler library captures summary message passing data while the trace library essentially timestamps MPI calls. The information can be conveniently displayed through the Peekperf tool.

Processor Performance [Slides 40-43]

We can obtain a variety of information on the performance a code on a processor through the use of the Xprofiler tool. This tool is much like gprof that many experienced code developer are use to. Using procedure profiling a graphical display of call graph can be produced. Hot spots in the code can be readily identified. Using the source code window and the disassembly windows users are able to identify to the line those parts of the codes that are the most CPU intensive. The dissembler code is useful in understanding results of various compiler options and quickly identifying the use of the double floating point unit. From this information, the user is able to easily revise his code in the higher language, recompile and gain improved performance on the processor.

Hardware Performance Monitors – Instrumented Code [Slides 44-46]

The hardware performance monitors are actually additional logic placed on the PowerPC processors. They count specific events and were originally intended for understanding of the logic by the processor designers. As a consequence, the additional logic is limited information available for an application developer. The hardware performance monitor library is the mechanism that applications developers can access the hardware performance monitor HPM. It requires the application developer to insert calls to the HPM library in his code. One advantage

of the HPM for the application developer is sections of code perhaps identified by the Xprofiler can be instrumented. The instrumentation can be nested to easily identify subsections. Again, the results of the HPM can be displayed through the Peekperf tool.

Running Code on Blue Gene/L [Slides 47-48]

When a user is ready to submit a run on Blue Gene/L, he first identifies and acquires a partition. This may be done through a script of a job scheduler or the user may allocate a partition depending on how the Blue Gene is administered locally. The allocation of a partition is done from the front-end node. Through the allocation, the user boots the partition to prepare it for execution of the program.

Multiple users may be using different partitions in the Blue Gene rack. This allows different users to share the rack but the individual partitions are not shared.

Once the partition is set up for the user, the user will submit his parallel job. The job submission may be done through a job scheduler or through the submit job command.

The `mpirun` command is also available for those familiar with it. The use of the **mpirun** command offers an advantage over the **submitjob** command because **mpirun** allows the allocation of the partition and the execution of the parallel job to be performed through a single command.

We present some results from two simple programs running on a Blue Gene/L system. All the results presented are from a partition consisting of 32 compute nodes. The information presented comes from editing the output on the front-end node. The results demonstrate a few features of the Blue Gene/L system.

The first program developed by Jim Sexton at Watson Research Center is a C code `C, Sanity.c` and the executable is referred to as `Sanity.rts`. It is a simple program that gathers during execution and subsequently prints out information on the compute node that the MPI task was running on. The information comes from the `bgpersonality.h`. The personality is static data given to every compute and I/O node at boot time. This simple code queries and extracts the information in the `bgpersonality` structure. The code was executed twice, once in coprocessor mode

and once in virtual node mode. Coprocessor mode is the default mode when booting a partition. If we were in coprocessor mode and wanted to run this program in virtual node mode, we would have to reboot the partition because of the static nature of the personality data. This is not a big task as the partitions on Blue Gene/L are design to quickly boot up.

There are two lines output from two separate runs, one representing coprocessor mode in the first line, VN? 0, and the other virtual node mode, where we have in the second line VN? 1. Both lines are line 20 from standard output file and the represent the 20th MPI rank which in line 1 is out of 32 MPI tasks (coprocessor mode 32 processors used) and in line 2 is out of 64 MPI tasks (virtual node mode 64 processors used). The next item prefaced by Pers indicates the X, Y, Z coordinates in the 4x4x2 mesh and the last coordinate indicating the T coordinate for the processors that share a compute node. The default torus ordering was assumed, X0Y0Z0. The Memory size is recorded to be 512 MB. The last item is the location of node in the system.

Line 1:

stdout[20]: MPI: 20/32, Pers: <0,1,1,0>/<4,4,2,1>, Torus? X0Y0Z0, VN? 0, Mem: 512MB(6), Loc: R00-M1-Nf-C:J14-U11

Line 2:

stdout[20]: MPI: 20/64, Pers: <0,1,1,0>/<4,4,2,2>, Torus? X0Y0Z0, VN? 1, Mem: 512MB(6), Loc: R00-M1-N2-C:J14-U11

The next program is a Monte Carlo code developed by Bob Walkup at Watson Research Center. It calculates the value of π using a different number of processors but keeping the amount of work the same. For this particular example, the code was running in coprocessor mode on a 32 compute node partition. In the results, it is of interest note the linear scaling, see the table below. In addition, note the time on each of the number of processors.

Monte Carlo Calculation of π

#cpus	#trials	pi(est)	err(est)	err(abs)	time(s)	Mtrials/s
32	256000000	3.14176	0.00022	0.00017	1.082	236.58
16	256000000	3.14164	0.00022	0.00004	2.164	118.29

8	256000000	3.14157	0.00022	0.00002	4.328	59.15
4	256000000	3.14160	0.00022	0.00000	8.656	29.57
2	256000000	3.14155	0.00022	0.00004	17.313	14.79
1	256000000	3.14145	0.00022	0.00014	34.625	7.39

APPLICATION PERFORMANCE

For an ultrascale out system with a massive number of processors, one might very well ask can one really take advantage of such a system on a single application. In addition, one might ask what problems/applications are suited to an ultrascale out architecture. In this section, we highlight a few applications that have some relevance to the computational biology community that have exploited the ultrascale out architecture of Blue Gene.

At IBM's Yorktown Heights facility, IBM runs a large Blue Gene system consisting of 20 Blue Gene racks. The purpose of this facility is for production science in support of IBM's Research Division's mission of basic science. This machine is referred to as BGW, Blue Gene at Watson.

Blue Matter Framework [Slides 50-54]

As part of the public unveiling of the Blue Gene project in December 1999, the project's two main goals were stated: (1) to advance our understanding of biological phenomena such as the mechanisms behind protein folding via large-scale simulation, and (2) to explore novel ideas in massively parallel machine architecture and software. This project should enable biomolecular simulations that are orders of magnitude larger than those achieved with previously available technology.

Blue Matter is a molecular simulation framework and application developed to support the scientific goals of IBM's Blue Gene project, to serve as a platform for research into application programming patterns for massively parallel architectures, and to explore ways to exploit hardware features of the Blue Gene/L architecture. A major design goal for Blue Matter has been to achieve strong scaling of molecular dynamics for moderately sized systems (10,000 – 100,000 particles) to very small numbers of atoms per node. This supports one of the aims of the scientific

component of the project, to carry out simulations on a scale that allows meaningful comparisons with experimental data. Results from early production use of prototype Blue Gene/L hardware were recently published in the Journal of the American Chemical Society.

Included here are results from the Blue Matter Framework for various molecular dynamic problems, such as G Protein-Coupled Receptors (GPCR) in a membrane environment and lipid bilayers. For comparison purposes, we provide the Blue Matter code performance against that of a popular free available code NAMD, the best NAMD performance. One key component of the Blue Matter code is to have an efficient 3D FFT. The performance of this kernel is displayed.

New Science and Outstanding Performance [Slides 55-63]

Several codes with some relevance to this community have achieved significant performance milestones. These are documented last fall at the SC05 conference, where the ddcMD code broke the 100 Tflops barrier. We describe the science and the performance results for ddcMD, a scalable, general purpose code for performing classical molecular dynamics (MD) simulations using the highly accurate model generalized pseudopotential theory (MGPT) potentials. These semi-empirical potentials, which are based on a rigorous expansion of many body terms in the total energy, are needed in order to investigate quantitatively the dynamic behavior of transitions metals and actinides under extreme conditions. What is shown is the nucleation of the solid phase in Tantalum and Uranium. This is the first time that it has been possible to see multiple nucleation sites in a computer modeling experiment. We also see that scaling of the code to greater than 10,000 processors is excellent.

Similarly, the CPMD code from IBM Zurich Research Labs has also exceeded the 100 Tflops mark. The CPMD is based on the Car-Parrinello Molecular Dynamics codes. The code uses a plane wave/pseudopotential implementation of the Density Functional Theory. The parallel implementation is designed for ab-initio molecular dynamics. For more detailed information including how to download the code visit: <http://www.cpmc.org> .

The work being done at AIST combines genome decoding with protein engineering for drug design. This is natural use of Blue Gene.

Not every computational biology problem is related to molecular dynamics. We mentioned the transcription regulatory work at Boston University. We will give details on the use of Blue Gene for genome assembly later. But another area the Blue Gene is impacting the biological sciences is the neurology. The Blue Brain Project at EPFL (Ecole Polytechnique Fédérale de Lausanne) in Switzerland headed by Prof. Henry Markham is using Blue Gene to accelerate their research efforts.

MASSIVELY PARALLEL COMPUTING ENVIRONMENT

Massively Parallel Computing Environment - What Next?

[Slide 64-72]

In the above, it was mentioned that a familiar programming model, Single Program Multiple Data (SPMD), is easily used on Blue Gene/L. Many cluster programs assume this programming paradigm. As a consequence, it is easy to get started on Blue Gene/L. In some instances, the programs moved to Blue Gene will tackle larger problems of interest to society with real impact. In other instances to get more accuracy, finer resolution, greater refinement or larger searches may be done utilizing a lot of processors. To take full advantage of the power of tens or even hundreds of thousands of processors, one may want to rethink the problem. For some problems, a Multiple Program Multiple Data (MPMD) paradigm may be more appropriate. The Blue Gene system can support such a model through use of MPI communicators. Through the use of multiple communicators, different parts of the Blue Gene system may be able to work on different aspects of the problem simultaneously. While this may not really be a Multiple Program Multiple Data situation, it provided the initial thinking in this direction.

In Biology, we are interested in exploring interactions of complex systems. We might model and perform simulations at various levels such as at the molecular, cellular, organ or organism level. But of real interest is to understand the interactions between these different levels. Even trying to fully model a single cell, a simple system, result in a set of complex systems interacting. A MPMD programming paradigm may be the right way to tackle such a problem.

Examples where concepts of multiple program with multiple data may apply abound in biology. Blue Gene provides the infrastructure with the ability to do atomistic

(discrete) level computations and easily couple these to microscale, mesoscale and continuum computations. One might look at the whole cell as composed of different simulations all coupled together or pharmaceutical manufacturing problems that involve multiple scales such as for inhalers.

Other examples include, analyzing the various data available from gene expression profiling, protein expression profiling, combined with multi-modal imaging and then coupled to models for simulation. Ultimately, as we learn how to use massively parallel computing environment to model biological systems, we will be able to rely on these models for predictions of outcomes. This eventually will lead to moving through the various biological scales, moving from gene data, to proteins, to cells and ultimately to organs and finally to complex biological systems. With systems like Blue Gene, we are starting to get the needed computational power but there remains a lot to learn on how to best use such tools combined with other tools such as high resolution imaging device. All of this is moving us in the realm of systems biology.

The goal of systems biology is to provide predictive simulations of complex biological problems. Here again we are faced with systems of complex systems. Modeling each system and then combining them in some fashion may lead naturally to a MPMD programming paradigm. Blue Gene/L provides the facilities to seriously explore this realm and provides the opportunity for new discoveries.

Large-Scale Computational Genomics on the IBM Blue Gene/L Supercomputer

We first present an overview of a number of large scale problems in computational genomics and then present a unified methodology to solve them on high performance parallel supercomputers such as the Blue Gene/L.

Genome Assembly [Slides 75-76]

Current sequencing techniques can only determine short sequences experimentally (up to 1000 nucleotides). To overcome this limitation, sequencing projects break a large sequence into many smaller sequences. As an example, consider whole genome sequencing. Multiple large genomic sequences are randomly fragmented

such that the output is a collection of unordered, but overlapping, fragments that can be sequenced. Genome assembly algorithms then take advantage of the inherent overlap information to reconstruct the original sequence similar to putting together a very large jigsaw puzzle.

Expressed Sequence Tag [Slides 77-78]

Expressed Sequence Tag (EST) sequencing is an approach used to enrich for the protein-coding sequences present in a genome. Messenger RNA is extracted from multiple tissues of interest, converted into cDNA via reverse transcription and cloned to generate a collection of expressed molecules often called a library. One or both ends of these cDNAs are then sequenced, resulting in many single-pass fragments. As shown in slide 5, however, genes are not uniformly expressed and in the absence of biochemical normalization the rate at which a specific gene is sampled is directly proportional to its frequency in the original cDNA pool. This frequency information is useful to analyze gene expression computationally, but also significantly affects the computational complexity because there are a quadratic number of potential overlaps among these data. For this reason most collections of ESTs are first grouped based on a single-linkage clustering algorithm to reduce their analysis to multiple subproblems.

Single Nucleotide Polymorphisms - SNP [Slides 79-82]

Although the underlying process used to generate genomic fragments and ESTs differ greatly, these sequencing strategies are highly complementary because ESTs can be mapped onto their respective genomic loci using algorithms collectively called *spliced alignment*. For example, it is possible to compute an optimal alignment solely based on scores assigned to matches, mismatches and constant penalties for insertions/deletions that satisfy a specified minimum length criterion. Moreover, ESTs are invaluable resources for locating single base differences between related individuals called single nucleotide polymorphisms, or SNPs. This is performed by analyzing a multiple sequence alignment composed of a reference sequence (e.g., an assembled genome) and multiple ESTs as illustrated in these slides. An alternative SNP detection approach is to perform an assembly of the diverse EST sequences themselves and determine columns of the corresponding multiple sequence alignment that differ, which is useful when no reference sequence is available.

Naïve Approach to Alignment

[Slide 83-85]

The above slides are an example of some of the problems in computational genomics which can in principle be solved by finding good alignments between pairs of sequences and processing the results. This tutorial will focus on how to solve such problems effectively on large parallel computers. The naïve approach to determine which sequences overlap involves computing an alignment between all pairs of input sequences; this is computation-intensive and often wasteful. Avoiding an all-vs.-all comparison is especially important for large scale data. For example, some of the largest EST collections such as human, mouse and maize are composed of millions of input ESTs. To overcome this pitfall, many software tools restrict the search space to sequences that are more likely to have a significant alignment by using criteria such as one or more exact matches of length l . Locating all sequences that share common exact matches can be performed using a linear time and space data structure such as a lookup table, which can then be used to restrict alignments to only between these “promising pairs”. Even so, ESTs may be non-uniformly sampled such that the number of promising pairs computed could scale quadratically even when using the most stringent of heuristics.

Lookup Tables

[Slide 86]

Lookup tables are easy to program, require linear space, and have a very small space constant, which enables them to be useful on many machines. There are, however, limitations with this approach. We would expect that the quality of an alignment is somewhat correlated with the longest exact match between two strings. Lookup tables can not determine the length of long exact matches because the space required to store them grows by a factor of 4^l for indexing substrings of length l . Moreover, a single long exact match of length m would reveal itself as $m - l + 1$ fixed length matches with no implicit order unless we initially sort all of the pairs.

PaCE Methodology

[Slide 87]

Our contribution, which is implemented as a software tool called *PaCE*, overcomes the limitations of a lookup table and introduces additional algorithmic innovations for large-scale sequence analysis. This software constructs a distributed version of a generalized suffix tree (GST) that allows generating promising pairs in decreasing

order of maximal common substring length in time and space proportional to the size of the input. In addition, an on-demand non-increasing pair generation scheme eliminates storing a potentially quadratic number of overlaps while decreasing the total amount of work by filtering these matches based on previously computed results. This provides a scalable framework that we show can effectively utilize thousands of processors to reduce run-time and increase available memory when processing large sequence collections.

Maize Genome Assembly

Maize Genome [Slide 88-89]

Here, we present an application of the PaCE framework to both maize genome assembly and large-scale EST clustering. Just as the human genome will accelerate advances in medicine, recent plant genome sequencing endeavors should help improve food production. Maize, also known as corn, is the best studied model for the cereal crops (rice, wheat, barley, and oat) and is itself economically important. Unfortunately, it is also one of the most complex eukaryotic genomes with a total size of ~2.5-3 billion bases, most of which is composed of highly similar retrotransposons. These present a significant challenge during assembly and were the focus of pilot studies to better understand the landscape of the maize genome.

Assembly Strategy [Slides 90-91]

As mentioned earlier, software tools that perform all pairwise comparisons such as genome assembly use an exact match filter to substantially reduce the search space. This also can lead to a linear space requirement in most shotgun sequencing projects because the number of overlaps each sequence participates should, on average, be the same as the number of times that base is represented in the set of sequences. Even so, assembly of genomes as large as the human genome and maize places enormous computational demands and requires tens of thousands of CPU hours. For example, twenty seven million fragments were assembled by the Celera Assembler in 20,000 CPU hours, half of which was spend detecting overlaps.

Maize Genome Assembly Effort [Slides 92-96]

Current estimates state that maize contains approximately 50,000 genes, which are roughly twice as many genes in the human genome and the model plant *Arabidopsis*. These genes, however, only occupy about 15-20% of the maize genome. To address this disparity and to devise a strategy for maize genome sequencing, an NSF workshop was held in 2001 to discuss various options and resulted in two consortiums being funded to explore multiple strategies. One consortium that involved Rutgers and the University of Arizona performed traditional Bacterial Artificial Chromosome (BAC) sequencing including a substantial number of end sequences to obtain a relatively random sampling of the maize genome at multiple resolutions: both at random (ends) and specific blocks (BACs). The other consortium involving the Danforth Plant Science Center, TIGR, Purdue, and Orion Genomics tested techniques that have since been termed “gene-enrichment” because they apply unique filters to traditional genomic fragments to substantially enrich for genic sequences. Each of these approaches is explained in detail below.

Arguably the best enrichment strategy for plant genomes involves selecting against methylated DNA prior to sequencing using special strains of *E. coli*. Unlike mammalian genomes, genes in plants tend to be highly hypomethylated, i.e., there are no methyl groups attached to the genome at these loci. Moreover, the repetitive sequences in plants tend to be highly methylated. Removing methylated DNA therefore enriches for genic sequences because repeats are preferentially removed during this strategy. This strategy is also very simple; only the type of bacteria used during the clone step needs to be changed in order for this strategy to work. The other gene-enrichment technique involved C₀T filtration using methodology similar to that used to normalize cDNA libraries prior to EST sequencing in the 1990s. Although this approach is also effective, it is much more complex and is subject to sequence artifacts because single-stranded DNA must be converted to double-stranded DNA prior to sequencing.

Cluster-Then-Assemble [Slides 97-98]

Gene-enrichment has been shown to be a cost-effective strategy to survey the “gene space” of at least two genomes: maize and sorghum. These strategies, however, place the same computation requirements on assembly algorithms as ESTs because certain sequences may preferentially survive the filtration steps

and/or residual repeats cause non-uniform sampling of certain regions of the genome.

To overcome the potential computational bottleneck of non-uniform genome assembly, we developed a “cluster-then-assemble” strategy. This strategy locates all of the connected components in the overlap graph used during genome assembly and then runs a serial assembly on each component in parallel. In order for this strategy to work, any overlap that would be determined by the assembler also should be detected during the clustering step to ensure the result of processing each connected component individually is the same as performing the assembly directly.

Our strategy proceeds as follows: We perform single-linkage clustering approach on a collection of gene-enriched sequences generated from methyl-filtration (MF) and/or high C₀T selection (HC). Specifically, each sequence belongs to its own cluster at the start of execution and at the end of clustering sequences belonging to the same cluster either directly overlap or are transitively related by a chain of overlaps that lead from one sequence to another. Because these chains may be inconsistent in the presence of repetitive sequences or chimeric reads, post-processing may generate more than one contig. We use the union-find data structure to efficiently combine and search for membership in clusters at any point during the execution of the algorithm.

Pair Generation [Slides 99-101]

One observation that can be made about clustering is that pairs that already belong to the same cluster do not need to be evaluated. In fact, it is easy to prove that although there are $O(n^2)$ potential overlaps in the worst case, only $O(n)$ of these can be used to merge clusters after which all n sequences will belong to the same cluster. Although we can not guarantee that we will find this linear number of successful alignments first, we can utilize a greedy clustering heuristic that attempts to find as many of these good overlaps quickly to eliminate many unsuccessful alignments. Our algorithm achieves this result by generating promising pairs in non-increasing order of maximal common substring length using a GST in amortized $O(1)$ time per pair. We do not need to store previously generated pairs, resulting in a linear space clustering algorithm for any underlying sequence sampling distribution.

Our PaCE algorithm has four distinct modules. In the first module, a generalized suffix tree, which is a compacted trie of all of the suffixes of all of the input strings, is constructed in parallel. There are currently no optimal suffix tree construction algorithms for distributed memory systems; however, observe that a suffix tree can be converted into a forest where each tree has a common prefix. In other words, we chop the suffix tree breadthwise at some arbitrary depth d generating at most 4^d subtrees whose suffixes share a common prefix. By setting the minimum allowable maximal common substring to be greater than d , we ensure that any promising pair can be determined on a single processor without utilizing any subsequent communication. To achieve this distributed GST, we initially bucket the suffixes based on the first d nucleotides and then allocate the corresponding buckets such that each processor has $O(nl/p)$ suffixes, where l is the average sequence length and p is the number of processors. Each subtree of the GST is then generated top-down by successive bucketing, resulting in an $O(n^2/p)$ algorithm, which works well in practice.

To eliminate additional memory overhead, we construct each subtree on each processor iteratively. Although this approach is space-efficient on the Blue Gene/L supercomputer, it requires substantial random access I/O in parallel and the same sequence may be read from disk as many times as it has suffixes. We addressed this issue on BG/L by replacing I/O with communication and using the collective memory of all the processors to store the input sequences.

Maximal common substrings between pairs of sequences can be identified as follows: The concatenation of edge labels on the path from root a node in the suffix tree is called the path label of that node. Consider an internal node in the tree and two leaves in its subtree. The path labels of these leaves correspond to two suffixes. The path label of the internal node is a common prefix to these suffixes, hence a shared substring. This common substring is right maximal (i.e., cannot be extended to the right) if the leaves are in the subtrees of different children of the internal node. They are left maximal (i.e., cannot be extended to the left) if the respective previous characters of the two suffixes differ. These rules are used to generate maximal

common substrings at every internal node in the tree. By sorting the internal nodes according to non-increasing order of the lengths of their path labels, and processing them in that order, the required pair generation is achieved.

Master-Worker Paradigm [Slide 115-117]

The clustering phase of the PaCE algorithm utilizes a master-worker paradigm where the master processor is responsible for cluster management and the worker processors generate pairs from their portion of the GST and compute alignments using banded dynamic programming. As shown in these slides, this approach scales well and, most importantly, is able to substantially reduce the number of alignments computed by processing exact matches in non-increasing length order.

Assembly Pipeline [Slide 118]

Our sequence assembly pipeline can be broken down as follows. First, input sequence data are cleaned by trimming residual vector sequence in addition to low-quality sequences using the Lucy tool. Then, sequences are masked using a modified version of the BLAST search algorithm using previously published methodology and clustered on BG/L. Each of the clusters is then assembled in parallel by distributing them to individual processors and combining the results after all clusters have been processed.

Blue Gene Performance [Slides 119-120]

We have applied our parallel framework on up to 8,192 processors of a BG/L supercomputer as illustrated by these tables. The implementation used for generating these runtime appears to scale well up until 8,192 processors for the suffix tree construction phase but only achieves marginal speedup during the clustering phase when compared to a run on a 1,024 processor system. This is primarily a result of using a single master processor for clustering that becomes overwhelmed as the number of processors enters the thousands. A potential solution to this phenomenon is to utilize multiple master processors; however, this may require substantial communication to ensure the state of the clusters is consistent between master processors.

Maize/Sorghum Assembled Genomic Islands [Slides 121-228]

The assembly results from our runs on BG/L are actively utilized by the plant biology community. We currently provide multiple assembly versions of both the gene-rich regions of the maize and sorghum genomes as builds called Maize Assembled Genomic Islands (MAGIs) and Sorghum Assembled Genomic Islands (SAMIs). The latest MAGI build (version 4.0) is based upon 3.2 million input fragments and is composed of 217,106 contigs and 567,797 non-repetitive sequences that do not assemble with any other sequence. Significantly, we have shown that the results of this assembly are highly accurate both with respect to base fidelity as well as contig formation in a 2005 paper in the Proceedings of the National Academy of Sciences that involved substantial biological validation. There are very few residual sequencing errors in our assembly and we estimate that overall this build has an error once every 10,000 nucleotides. This assembly has led to substantial advances in maize genetics and one such observation is that genes tend to cluster together in the maize genome in what we term gene “archipelagoes”. These data in addition to information about the PaCE software tool can be accessed from our project website <http://magi.plantgenomics.iastate.edu>. This site also includes substantial annotation graphically displayed by GBrowse and is searchable by BLAST.

Maize Genome Project Future [Slides 129-130]

The U.S. National Science Foundation (NSF), Department of Agriculture (USDA), and Department of Energy (DOE) have recently announced a \$32 million dollar investment in maize genome sequencing. The goal of this project is to sequence all genes in the B73 cultivar, determine their order and orientation, and anchor them to the genetic/physical maps. Washington University in St. Louis will lead this sequencing consortium that includes the University of Arizona, Iowa State University and Cold Spring Harbor.

Mouse EST Clustering

Mouse EST [Slides 131-132]

We have also applied our clustering framework to the ~3.8 million mouse EST sequences, which is the second largest collection of ESTs next to those from human. To validate our massively parallel strategy we reclustered sequences from

a UniGene build downloaded in March 2006 on 1,024 processors of BG/L and obtained 60,862 clusters with more than one sequence; the original UniGene collection contained 56,470 clusters. Approximately 83% of these clusters are composed of members obtained from a single UniGene without postprocessing the results to eliminate highly similar gene families and/or alternative splice forms.

Validating Accuracy

[Slides 133-134]

Validating the accuracy of any clustering algorithm can be performed in different ways. One method is based on all possible pairs of sequences, and this approach has been used to test the ability of PaCE to cluster small collections of plant ESTs. Another approach that we found yielded more information for large collections of ESTs is comparing the number of correct merges, or decisions, computed by our algorithm when compared to the UniGene benchmark. Similarly, we can determine how many additional merges PaCE performed (false positives) that led to two different UniGenes being placed together as well as the number of merges missed (false negatives) leading to multiple PaCE clusters per UniGene cluster. As illustrated on slide 61, over 3.2 million out of 3.3 million decisions were made by both clustering algorithms, suggesting PaCE recovers many of the valid overlaps among the maize ESTs processed. Moreover, only 26,125 false positive merges were made, confirming that many clusters are highly specific.

Blue Gene Performance and Scaling

[Slides 135-136]

The scaling of our BG/L clustering algorithm when applied to EST data is similar to that of maize gene-enriched genomic fragments with near perfect scaling up to 1,024 processors for the largest dataset analyzed ($n = 2,000,000$) and weak scaling for smaller datasets. Interestingly, the alignments saved when performing EST clustering is substantially higher for these sequence data than genomic data with nearly 1.8 billion alignments ignored for the 3.78 million dataset among a total 2.1 billion (86%). Even so, this run took a total of 9.5 hours, of which 7 hours were spent clustering.

Closing Remarks

Blue Gene Consortium

[Slide 139]

There are several places to learn more about how to use Blue Gene and how you might access it. One such place is to visit the Blue Gene Consortium Website, <http://www.mcs.anl.gov/bgconsortium> . One can learn about other activities at this Website.

Final Remarks

[Slide 140]

While we are starting to see Blue Gene impacting computational biology, much work remains to be done. In closing, it will be up to computational biology community to determine how a resource like Blue Gene will really have impact. Hopefully, this tutorial sparked your interest and your imagination on problems you might now tackle using a resource such as Blue Gene.

REFERENCE LIST/ADDITIONAL READING

Adiga N. R., et al, "An Overview of The Blue Gene/L Supercomputer", IBM Journal of Research and Development, Volume 40, Number 2, 2001, p. 310

Adiga N. R. et al. Blue Gene/L torus interconnection network. *IBM Journal of Research and Development*, 49(2/3):265–276, 2005

Allen, F. et al. Blue Gene: a vision for protein science using a petaflop supercomputer. *IBM Systems Journal*, 40(2):310–327, 2001.

Allsopp, N. et. al., Unfolding IBM eServer Blue Gene Solution, *IBM Redbooks* (<http://www.ibm.com/redbooks/>), September 2005.

Almasi G. et al , "Unlocking the Performance of the BlueGene/L Supercomputer", IEEE Comp. Soc. 57 2004.

Almasi G et al. Design and implementation of message-passing services for the Blue Gene/L supercomputer. *IBM Journal of Research and Development*, 49(2/3):393–406, 2005.

Bhanot, G, J. M. Dennis , J. Edwards , W. Grabowski, M. Gupta , K. Jordan, R. D. Loft , J. Sexton , A. St-Cyr , S. J. Thomas, H. M. Tufo , T. Voran, R. Walkup , and A. A. Wyszogrodski, " An Atmospheric General Circulation Model for BG/L", 2005 Gordon Bell Award submission (Special Category).

Bhanot G., D. Chen, A.Gara, J. C. Sexton, P.Vranas, 2005 Gordon Bell Award submission (Special Category)

Bhanot G., A. Gara P. Heidelberger E. Lawless, J. C. Sexton, R. Walkup, "Optimizing task layout on the Blue Gene/L Supercomputer" IBM Journal of Research and Development, Vol 49, No. 2/3, March/May, 2005.

Boczko, E.M. and C.L. Brooks. First-Principles Calculation of the Folding Free-Energy of a 3-Helix Bundle Protein. *Science*, 269(5222): 393-396, 1995

Duan, Y. and P.A. Kollman. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282: 740, 1998.

- Deserno, M. and Holm, C. How to mesh up ewald sums. i. a theoretical and numerical comparison of various particle mesh routines. *J. Chem. Phys.*, 109(18):7678–7693, 1998.
- Dobson, C.M., Protein folding and its links with human disease, in *From Protein Folding to New Enzymes*. 2001. p. 1-26.
- Eleftheriou, M., et al. Scalable framework for 3d FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements. *IBM Journal of Research and Development*, 49(2/3):457–464, 2005.
- Fitch, B.G. et al. Blue Matter, an application framework for molecular simulation on Blue Gene. *Journal of Parallel and Distributed Computing*, 63:759–773, 2003.
- Gara, A. et al. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2/3):195–212, 2005.
- Germain, R.S. et al. Early performance data on the Blue Matter molecular simulation framework. *IBM Journal of Research and Development*, 49(2/3):447–456, 2005.
- Giampapa, M.E. et al. Blue Gene/L advanced diagnostics environment. *IBM Journal of Research and Development*, 49(2/3):319–332, 2005.
- Hansmann, U.H.E. Parallel tempering algorithm for conformational studies of biological molecules. *Chemical Physics Letters*, 281(1-3): 140-150, 1997.
- Lakner, G., Mullen-Schultz, G.L., Barnard, E. W, Blue Gene/L: System Administration, *IBM Redbooks* (<http://www.ibm.com/redbooks/>), Draft June 2006.
- McCammon, J.A., B.R. Gelin, and M. Karplus. Dynamics of folded proteins. *Nature*, 267(5612): 585-590, 1977
- Mullen-Schultz, G.L., Blue Gene/L: Application Development, *IBM Redbooks* (<http://www.ibm.com/redbooks/>), Draft June 2006.
- Mullen-Schultz, G.L., Blue Gene/L: Performance Analysis Tools, *IBM Redbooks* (<http://www.ibm.com/redbooks/>), Draft June 2006.

- Munoz, V., et al. Folding dynamics and mechanism of beta-hairpin formation. *Nature*, 390: 196-198, 1997.
- Nyland, L., et al. Achieving scalable parallel molecular dynamics using dynamic spatial decomposition techniques. *Journal of Parallel and Distributed Computing*, 47(2):125–138, December 1997.
- Phillips, J.C., et al. NAMD: biomolecular simulation on thousands of processors. In *Supercomputing 2002 Proceedings*, 2002.
<http://www.sc2002.org/paperpdfs/pap.pap277.pdf>.
- Pitman, M.C., et al. Molecular-Level Organization of Saturated and Polyunsaturated Fatty Acids in a Phosphatidylcholine Bilayer Containing Cholesterol. *Biochemistry*, 43(49): 15318-15328, 2004.
- Pitman, M.C., et al. Role of Cholesterol and Polyunsaturated Chains in Lipid-Protein Interactions: Molecular Dynamics Simulation of Rhodopsin in a Realistic Membrane Environment *J. Am. Chem. Soc.* 127(13) : 4576 – 4577, 2005 DOI: 10.1021/ja042715y
- Plimpton, S. and Hendrickson, B. A new parallel method for molecular dynamics simulation of macromolecular systems. *Journal of Computational Chemistry*, 17(3):326–337, 1996.
- Shaw, D.E. An asymptotic improvement in the parallel evaluation of pairwise particle interactions. *Presented at Philadelphia American Chemical Society meeting*, September 2004.
- Sheinerman, F.B. and C.L. Brooks III. Molecular picture of folding of a small alpha/beta protein. *PNAS*, 95: 1562-1567, 1998.
- Simmerling, C., B. Strockbine, and A.E. Roitberg. All-atom structure prediction and folding simulations of a stable protein. *Journal of the American Chemical Society*, 124(38): 11258-11259, 2002.
- Snir, M. A note on n-body computations with cutoffs. *Theory of Computing Systems*, 37:295–318, 2004.

Snow, C.D., et al. Absolute comparison of simulated and experimental protein-folding dynamics. *Nature*, 420(6911): 102-106, 2002.

Straatsma, T.P. and McCammon, J.A. Load balancing of molecular dynamics simulation with NWChem. *IBM Systems Journal*, 40(2):328–341, 2001.

Suits, F. et al. Overview of molecular dynamics techniques and early scientific results from the Blue Gene project. *IBM Journal of Research and Development*, 49(2/3):475–488, 2005.

Swope, W.C. et al. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *Journal of Chemical Physics*, 76:637–649, 1982.

Swope, W.C., J.W. Pitera, and F. Suits. Describing protein folding kinetics by molecular dynamics simulations. 1. Theory. *Journal of Physical Chemistry B*, 108(21): 6571-6581, 2004

Swope, W.C., et al. Describing protein folding kinetics by molecular dynamics simulations. 2. Example applications to alanine dipeptide and beta-hairpin peptide. *Journal of Physical Chemistry B*, 108(21): 6582-6594, 2004.

Top500 Supercomputer Sites, 28th Top500 List, (<http://www.top500.org/>), June 2006.

Zhou, R.H., B.J. Berne, and R. Germain. The free energy landscape for beta hairpin folding in explicit water. *Proc. Natl. Acad. Sci. USA*, 98(26): 14931-14936, 2001.

Additional WEB Sites

Here are some additional websites, in addition those included in the text, where one can find additional information on the topics described in this tutorial.

IBM Deep Computing website contains other information on Blue Gene. The Website is:

<http://www.ibm.com/servers/deepcomputing/>

Information on the IBM Computational Biology Center can be found at:

<http://domino.research.ibm.com/comm/research.nsf/pages/r.compbio.html>

IBM's Healthcare and Life Sciences Industry details can be found at the following;

<http://www.ibm.com/industries/healthcare/>

IBM Redbooks are handy references guides on many topics. All IBM Redbooks are available as pdf files. The IBM redbook website is:

<http://www.ibm.com/redbooks>

Information on the Bioinformatics and Computational Biology Program at Iowa State is available at:

<http://www.bcb.iastate.edu/>

Information on the Department of Biomedical Engineering at Boston University is available at:

<http://www.bu.edu/dbin/bme>

Srinivas Aluru's home page is:

<http://www.ee.iastate.edu/~aluru/home.html>

Kirk Jordan's home page is:

<http://www.ibm.com/software/info/university/people/kjordan.html>

14th Annual International Conference On Intelligent Systems For Molecular Biology

I/MB 2006 Fortaleza, Brazil
August 6-10, 2006
and 2nd Annual AB³C Conference: X-Meeting

Exploring Computational Biology with a Massively Parallel High Performance Computing Environment

Kirk E. Jordan
IBM, USA
kjordan@us.ibm.com
Srinivas Aluru
Iowa State University, USA
aluru@iastate.edu

14th Annual International Conference On Intelligent Systems For Molecular Biology

Outline

- Background
- Brief overview example
 - Transcription Factor Binding Site Identification
- Architecture of a massively parallel environment
 - Hardware & Software Philosophy
- Performance Tools & Compilers
- Application Performance
- What might we do?
- Large-Scale Computational Genomics
- Specific Applications:
 - Maize Genome Assembly
 - Mouse EST Clustering
- Remarks

I/MB 2006 Fortaleza, Brazil
August 6-10, 2006

178

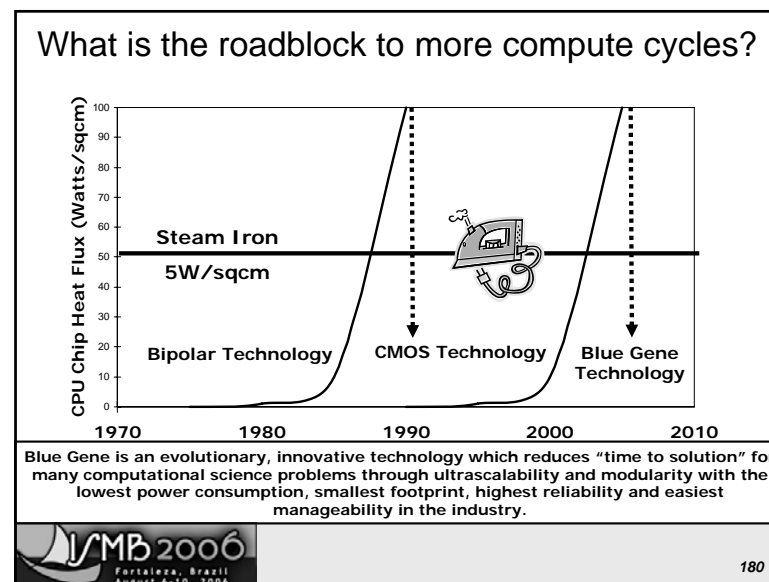
14th Annual International Conference On Intelligent Systems For Molecular Biology

I/MB 2006 Fortaleza, Brazil
August 6-10, 2006
and 2nd Annual AB³C Conference: X-Meeting

Background

Trends in parallel computing

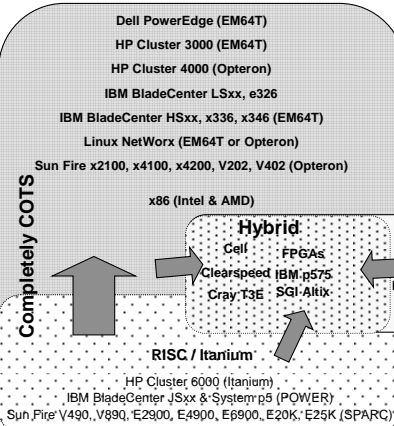
14th Annual International Conference On Intelligent Systems For Molecular Biology



Trends in HPC Architecture

Completely COTS

- Commercial off-the-shelf (COTS) processors
- Standard packaging (rack-optimized or blade)
- PCI-Express or HTX-based interconnects
- Often vendor integrated
- Largest & fastest growing segment
 - Best price/perf.
 - "good enough" technology



Hybrid

- Combines commodity processors with custom high speed interconnect and/or accelerators to enhance performance
- Standard or custom packaging
- Growth limited by program model difficulties & lack of enablement tools

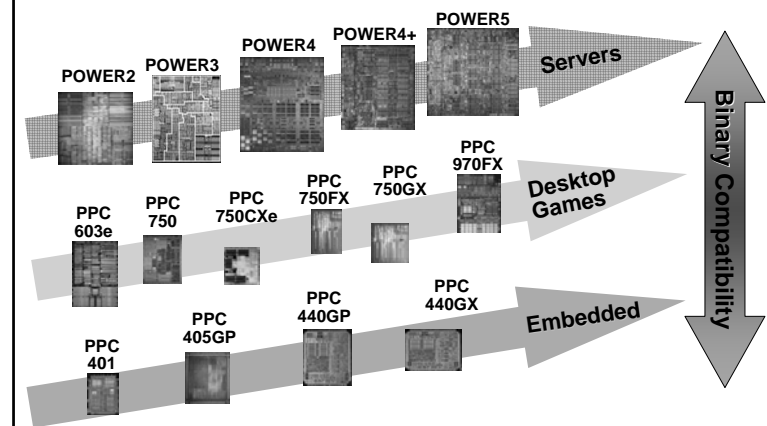
Custom

- Specialized processors, interconnects & processor-memory interfaces



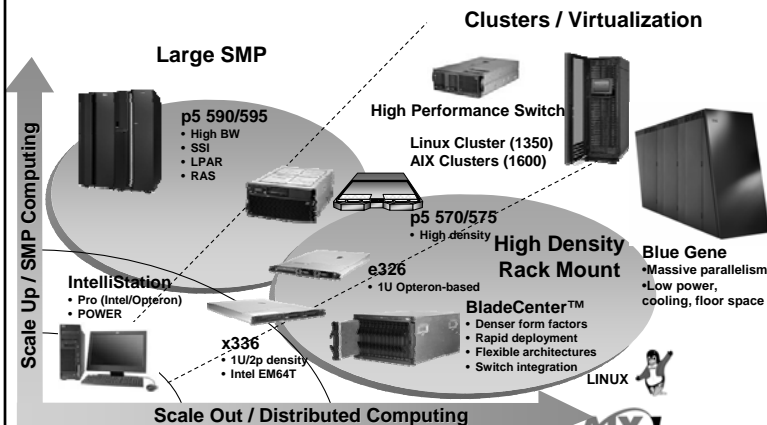
181

POWER : The Most Scalable Architecture



182

IBM Systems – Industry Leadership and Choice



183

IBM HPC Server Portfolio

Scale-out (high-value) <ul style="list-style-type: none"> Tightly coupled clusters RISC- or Itanium-based SMP servers; Optional high performance interconnect (industry-standard, OEM or custom) Industry standard or custom packaging Vendor integrated <p>System p5 575 IBM BladeCenter® JS21*</p>	Purpose-built <ul style="list-style-type: none"> Specifically designed for HPC capability workloads Usually custom microprocessors, usually employ vectors and streaming Custom interconnect Custom packaging Vendor integrated <p>IBM System Blue Gene® Solution</p>
Scale-out (commodity) <ul style="list-style-type: none"> Clusters of 1, 2, and 4-way blade or rack-optimized servers Based on "merchant" or low-cost technology Standard or OEM high performance interconnects & graphics adapters Standard packaging Broad ISV support with concomitant availability (Linux and Windows) Often vendor integrated <p>IBM BladeCenter® HS20 & LS20* IBM System x™</p>	(Standalone) SMP <ul style="list-style-type: none"> 2-way to 64-CPU (or bigger) SMP servers Single system simplicity, uniform memory programming model, and high SMP scalability for a broad range of problem sizes and throughput objectives Broad ISV support (Unix and Linux) <p>IBM System p5™ IBM System x™</p>



*Cluster 1350 Building Blocks

184



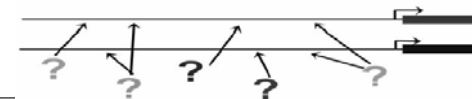
Example Using Blue Gene

Biology of Transcriptional Regulation

- Transcription Factors (TFs) bind DNA upstream of a gene and promote or inhibit RNA transcription
- Genes bound by the same TF can be co-regulated

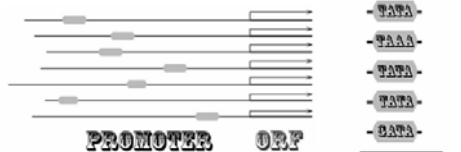
Goal

- Identify both the TFs and the places they bind (i.e. the genes they regulate)
- Identify sets of gene regulated by the same TF



Gibbs Sampling

- Pick initial positions in promoter hypothesized to contain a common binding site



- Calculate a PWM

- In general, the score we want to optimize is the conservation of the PWM

- Iteratively update positions, optimizing PWM



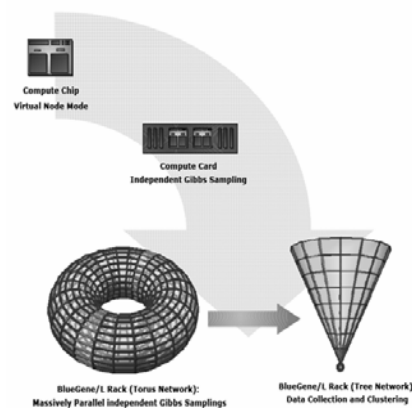
GibTigs BlueGene/L Implementation

- GibTigs analysis requires the compilation of **millions** of Gibbs sampling iterations.

- Each iteration is **independent**, and allowing broad distribution of jobs with minimal interprocessor communication

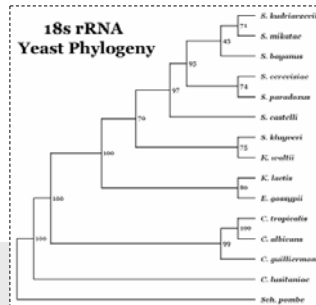
- GibTigs also takes full advantage of both the tree and torus networks connecting processors.

- Thus, **linear scalability** is maintained through the use of thousands of CPUs.



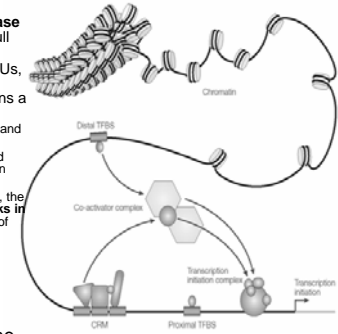
GibTigs on the *S. cerevisiae* genome

- **Study set:**
Genome location data for 203 TFs in numerous conditions.
310 sets of genes to be analyzed in all.
- Using a single BlueGene/L rack, completed whole genome GibTig analysis in less than 2 weeks.
- Potential for comparative genomic studies?
(genomic sequences exist for 15 yeast species)
- Will be possible with multi-rack BlueGene/L systems
(16 racks ~ 1 yeast genome per day)



Blue Gene/L - Insight into Genomic Regulation

- Redeployment of GibTigs on BlueGene/L systems redefines Boston University's Biomolecular Systems Lab development cycle.
 - One BG/L rack represents nearly a **twenty fold increase** in available compute power, of which we have taken full advantage.
 - GibTigs showing linear scalability up through 2048 CPUs, one BG/L rack in virtual-node mode - sped our development cycle from a few runs a week to many runs a day.
 - Doing so has **enabled large scale parameter searches**, and regular production grade performance evaluations.
 - The results are **drastic improvements in sensitivity** and specificity of our algorithm, none of which would have been possible at our previous development pace.
 - Rather than making conservative modifications to GibTigs, the power of BlueGene has given us the **freedom to take risks in trying new ideas**, many of which have failed, but some of which have **provided new insight and new power** to GibTigs.
 - **GibTigs has recently proved to be, according to published measures, the most powerful predictor of DNA transcription regulatory sites to date.**
- “Having achieved a performance milestone, the scalability of BlueGene/L has encouraged us to think even bigger in our research.”



190

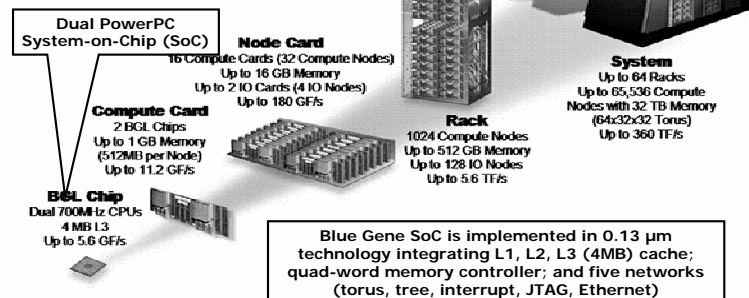
14th Annual International Conference On Intelligent Systems For Molecular Biology



Blue Gene Architecture

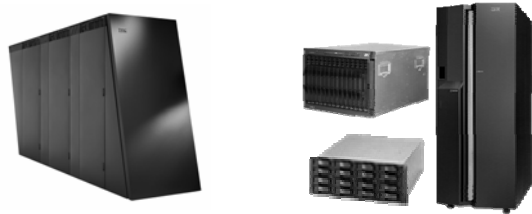
Hardware
Software

What is Blue Gene?



192

Blue Gene system modularity

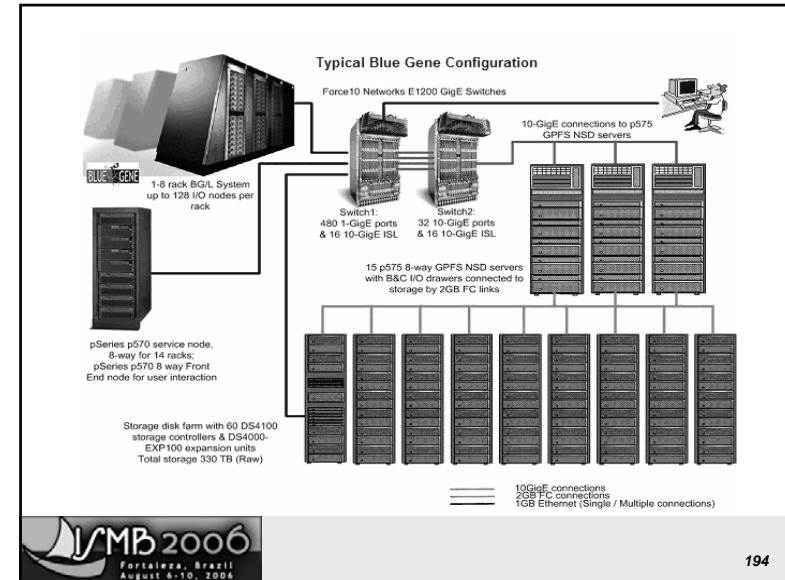


Blue Gene Rack(s)
Up to 1024 Compute Nodes / Rack
Up to 128 IO Nodes / Rack

Host System
Service and Front End Nodes
(P5/SLES9), Storage System,
Ethernet Switch, Cabling, SuSE
SLES9, DB2, XLF/C Compilers

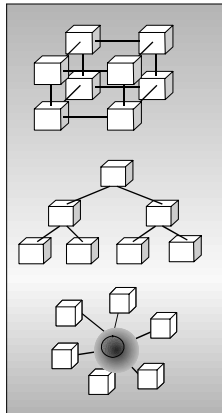


193



194

BlueGene/L Interconnection Networks



3 Dimensional Torus

- Interconnects all compute nodes (65,536)
- Virtual cut-through hardware routing
- 1.4Gb/s on all 12 node links (2.1 GB/s per node)
- Communications backbone for computations
- 0.7/1.4 TB/s bisection bandwidth, 67TB/s total bandwidth

Global Tree

- One-to-all broadcast functionality
- Reduction operations functionality
- 2.8 Gb/s of bandwidth per link
- Latency of tree traversal 2.5 μ s
- ~23TB/s total binary tree bandwidth (64k machine)
- Interconnects all compute and I/O nodes (1024)

Ethernet

- Incorporated into every node ASIC
- Active in the I/O nodes (1:64)
- All external comm. (file I/O, control, user interaction, etc.)

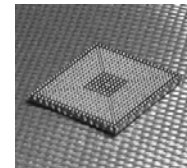
Low Latency Global Barrier and Interrupt

Control Network

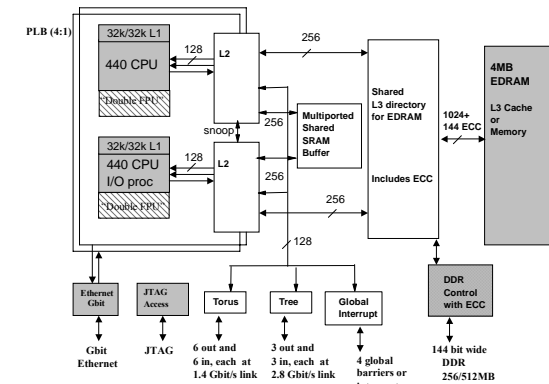


195

BlueGene/L Compute ASIC



- IBM CU-11, 0.13 μ m
- 11 x 11 mm die size
- 25 x 32 mm CBGA
- 474 pins, 328 signal
- 1.5/2.5 Volt

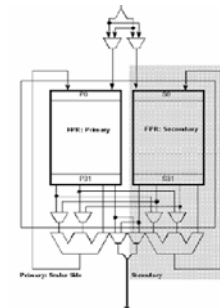


196

Powerpc-440 Processor

- 32-bit architecture at 700 MHz
- single integer unit (fxu)
- single load/store unit
- special double floating-point unit (dfpu)
- L1 Data cache : 32 KB total size, 32-Byte line size, 64-way associative, round-robin replacement
- L2 Data cache : prefetch buffer, holds 16 128-byte lines
- L3 Data cache : 4 MB, ~35 cycles latency, on-chip
- Memory : 512 MB DDR at 350 MHz, ~85 cycles latency
- Double FPU has 32 primary floating-point registers, 32 secondary floating-point registers, and supports :
 - standard powerpc instructions, which execute on fpu0 (fadd, fmadd, fadds, fdiv, ...), and
 - SIMD instructions for 64-bit floating-point numbers (fpadd, fpmadd, fpre, ...)
- Floating-point pipeline : 5 cycles
- Floating-point load-to-use latency : 4 cycles

Dual FPU Architecture



- Two 64 bit floating point units
- Designed with input from compiler and library developers
- SIMD instructions over both register files
 - FMA operations over double precision data
 - More general operations available with cross and replicated operands
 - Useful for complex arithmetic, matrix multiply, FFT
- Parallel (quadword) loads/stores
 - Fastest way to transfer data between processors and memory
 - Data needs to be 16-byte aligned
 - Load/store with swap order available
 - Useful for matrix transpose

Double FPU Code

For double FPU code generation, 16-byte alignment is required; should work from -qarch=440d but may need alignment assertions:

Fortran :

```
call alignx(16,x(1))
call alignx(16,y(1))
!ibm* unroll(10)
do i = 1, n
    y(i) = a*x(i) + y(i)
end do
```

C :

```
double *x, *y;
#pragma disjoint (*x, *y)
__alignx(16,x);
__alignx(16,y);
#pragma unroll(10)
for (i=0; i<n; i++) y[i] = a*x[i] + y[i];
```

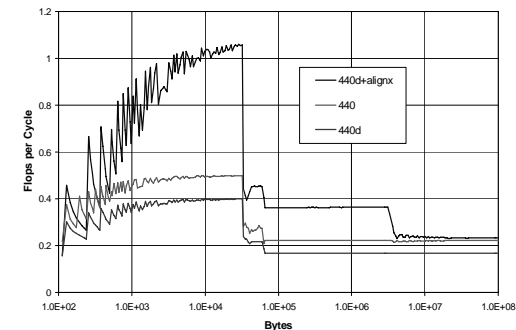
Try : -O3 -qarch=440d -qlist -qsource

Easiest approach to double FPU is to use optimized math library routines.

Alignment Performance

BG/L Daxpy Performance

```
call alignx(16,x(1))
call alignx(16,y(1))
do i = 1, n
    y(i) = a*x(i) + y(i)
end do
```



Performance of compiler-generated code is shown.

-qarch=440 => single FPU code, theoretical limit is 2/3 flops per cycle.

-qarch=440d => double FPU code, theoretical limit is 4/3

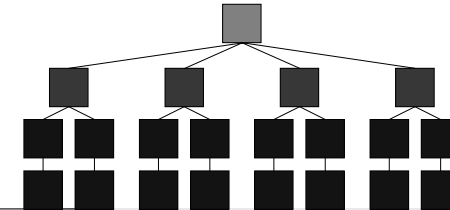
flops per cycle, data in-cache, 2/3 flops per cycle otherwise

The Software Solution Philosophy

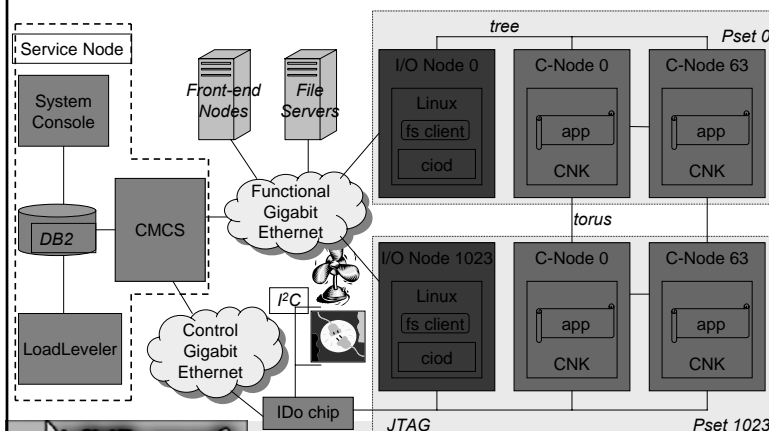
- **Simplicity**
 - Avoid features not absolutely necessary for high performance computing
 - Using simplicity to achieve both efficiency and reliability
- **New organization of familiar functionality**
 - Same interface, new implementation
 - Hierarchical organization
 - Message passing provides foundation
 - Research on higher level programming models using that base

BlueGene/L Software Hierarchical Organization

- Compute nodes dedicated to running user application, 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, signaling, debugging, and termination
- Service node performs system management services (e.g., heart beating, monitoring errors) - transparent to application software



BlueGene/L System Architecture



Programming Models and Development Environment

- **Familiar Aspects**
 - SPMD model - Fortran, C, C++ with MPI (MPI1 + subset of MPI2)
 - Full language support
 - Automatic SIMD FPU exploitation
 - Linux development environment
 - User interacts with system through FE nodes running Linux – compilation, job submission, debugging
 - Compute Node Kernel provides look and feel of a Linux environment – POSIX system calls (with restrictions)
 - Tools – support for debuggers (Etnus TotalView), MPI tracer, profiler, hardware performance monitors, visualizer (HPC Toolkit, Paraver, Kojak)
- **Restrictions (lead to significant scalability benefits)**
 - Strictly space sharing - one parallel job (user) per partition of machine, one process per processor of compute node
 - Virtual memory constrained to physical memory size
 - Implies no demand paging, only static linking
- **Other Issues:** Mapping of applications to torus topology
 - More important for larger systems (multi-rack systems)
 - Working on techniques to provide transparent support

Execution Modes for Compute Node

- **Communication coprocessor mode:** CPU 0 executes user application while CPU 1 handles communications
 - Preferred mode of operation for communication-intensive and memory bandwidth intensive codes
 - Requires coordination between CPUs, which is handled in libraries
 - **Computation offload feature (optional):** CPU 1 also executes some parts of user application offloaded by CPU 0
 - Can be selectively used for compute-bound parallel regions
 - Asynchronous co-routine model (co_start / co_join)
 - Need careful sequence of cache line flush, invalidate, and copy operations to deal with lack of L1 cache coherence in hardware
- **Virtual node mode:** CPU0 and CPU1 handle both computation and communication
 - Two MPI processes on each node, one bound to each processor
 - Distributed memory semantics – lack of L1 coherence not a problem



Performance

Compilers
Libraries
Tools
Running

HPC Tools Available for Blue Gene

IBM Software Stack

- XL Compilers
 - Externals preserved
 - New options to optimize for specific Blue Gene functions
- LoadLeveler
 - Same externals for job submission and system query functions
 - Backfill scheduling to achieve maximum system utilization
- GPFS
 - Provides high performance file access, as in current pSeries and xSeries clusters
 - Runs on IO nodes and disk servers
- ESSL/MASSV
 - Optimization library and intrinsics for better application performance
 - Serial Static Library supporting 32-bit applications
 - Callable from FORTRAN, C, and C++

Other Software

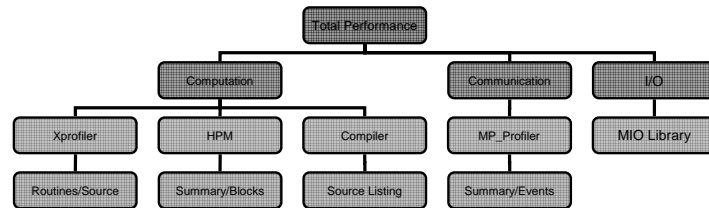
- Etnus TotalView
 - Parallel Debugger
- Lustre File System
 - Enablement underway at LLNL
- FFT Library
 - FFTW Tuned functions by TU-Vienna
- Performance Tools
 - Total View
 - HPC Toolkit
 - Paraver
 - Kojak

Scalar and Vector MASS Routines

Approximate cycle-counts per evaluation on BGL processor

	libm.a	libmass.a	libmassv.a
exp	185	64	22
log	320	80	25
pow	460	176	29 - 48
sqrt	106	46	8-10
rsqrt	136	...	6-7
1/x	30	...	4-5

Performance Decision Tree



Using IBM XL Compilers – Switches

Optimization levels:

Default optimization = none (very slow)

-O : good place to start, use with -qmaxmem=64000

-O2: same as -O

-O3 -qstrict : less aggressive, must strictly obey program semantics

-O3: aggressive, allows re-association, will replace division by multiplication with the inverse

-qhot : turns on high-order transformation module, will add vector routines, unless -qhot=novector

check listing: -qreport=hotlist

-qipa : inter-procedure analysis; many suboptions such as: -qipa=level=2

Architecture flags:

-qarch=440 : generates standard powerpc floating-point code

-qarch=440d : will try to generate double FPU code

Recommendation:

On BG/L start with : -g -O -qarch=440 -qmaxmem=64000

Try : -O3 -qarch=440/440d

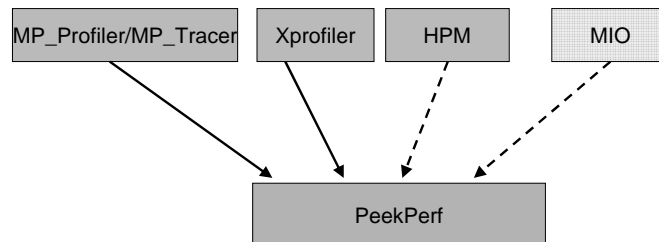
Try : -O5 -qarch=440d

-O4 = -O3 -qhot -qipa=level=1 -qarch=auto

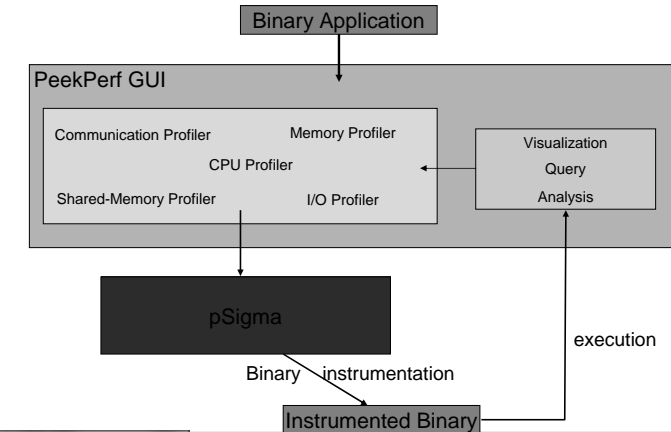
-O5 = -O3 -qhot -qipa=level=2 -qarch=auto

IBM High Performance Computing Toolkit on BG/L

- MPI performance: MP_Profiler, MP_Tracer
- CPU performance: Xprofiler, HPM
- Visualization and analysis: PeekPerf
- Modular I/O: MIO



Structure of the HPC toolkit



Message-Passing Performance:

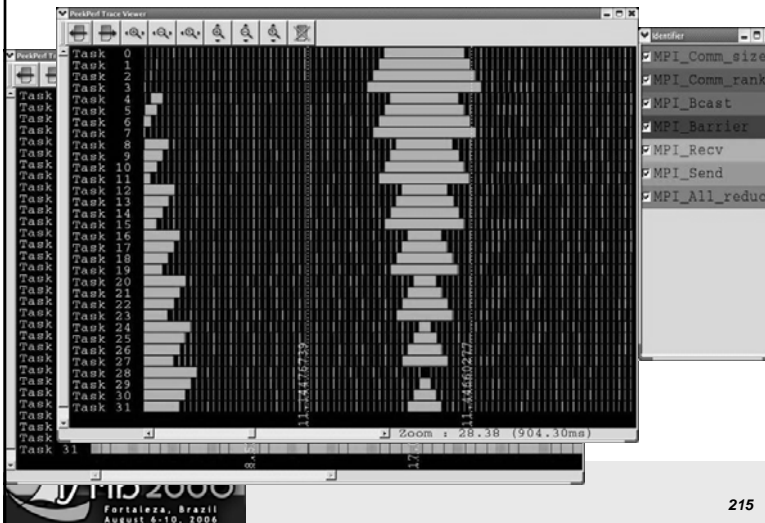
- MP_Profiler Library
 - Captures “summary” data for MPI calls
 - Source code traceback
 - User **MUST** call MPI_Finalize() in order to get output files.
 - No changes to source code
 - MUST compile with -g to obtain source line number information
- MP_Tracer Library
 - Captures “timestamped” data for MPI calls
 - Source traceback

MP_Profiler Output with Peekperf

The screenshot shows the MP_Profiler output window. It displays a list of MPI calls on the left, including barrier_sync, broadcast_int, broadcast_real, global_int_sum, global_real_max, global_real_sum, mpi_allreduce, mpi_barrier, mpi_bcast, mpi_comm_rank, mpi_comm_size, mpi_recv, mpi_send, rcv_real, and rcv_real. The right pane shows a table of MPI operations with columns: Task, Message Size, Count, WallClock (Max), Transferred Bytes, Call Count (Max), and WallClock. The table lists various MPI operations and their performance metrics.

Task	Message Size	Count	WallClock (Max)	Transferred Bytes	Call Count (Max)	WallClock
0	16K ... 64K	1820	0.512883	2.21184e+07	1820	0.512883
1	16K ... 64K	2880	0.549085	3.31776e+07	2880	0.549085
2	16K ... 64K	2880	0.551296	3.31776e+07	2880	0.551296
3	16K ... 64K	1820	0.516694	2.21184e+07	1820	0.516694
4	16K ... 64K	2880	0.532482	3.31776e+07	2880	0.532482
5	16K ... 64K	3840	0.554542	4.42558e+07	3840	0.554542
6	16K ... 64K	3840	0.551433	4.42558e+07	3840	0.551433
7	16K ... 64K	2880	0.525413	3.31776e+07	2880	0.525413
8	16K ... 64K	2880	0.550983	3.31776e+07	2880	0.550983
9	16K ... 64K	3840	0.543496	4.42558e+07	3840	0.543496
10	16K ... 64K	3840	0.543881	4.42558e+07	3840	0.543881
11	16K ... 64K	2880	0.580382	3.31776e+07	2880	0.580382
12	16K ... 64K	2880	0.553126	3.31776e+07	2880	0.553126

MP_Profiler - Traces

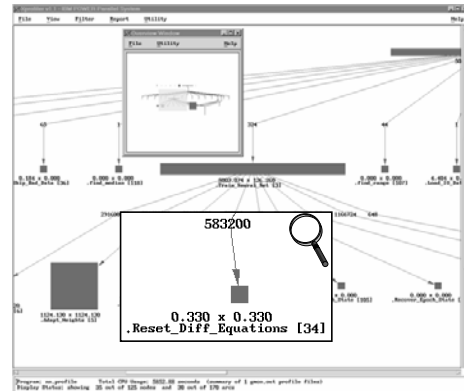


Xprofiler

- CPU profiling tool similar to gprof
- Can be used to profile both serial and parallel applications
- Use procedure-profiling information to construct a graphical display of the functions within an application
- Provide quick access to the profiled data and helps users identify functions that are the most CPU-intensive
- Based on sampling (support from both compiler and kernel)
- Charge execution time to source lines and show disassembly code

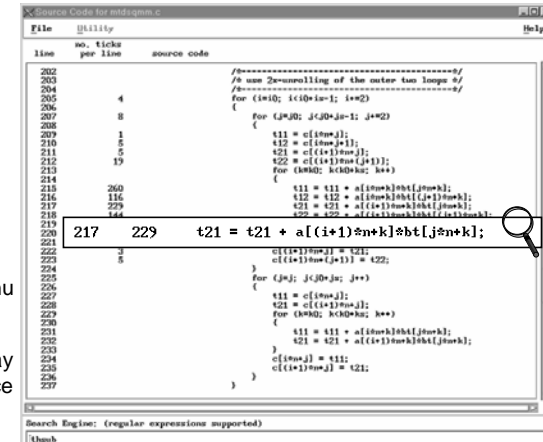
Xprofiler: Main Display

- Width of a bar: time including called routines
- Height of a bar: time excluding called routines
- Call arrows labeled with number of calls
- Overview window for easy navigation (View → Overview)



Xprofiler: Source Code Window

- Source code window displays source code with time profile (in ticks=.01 sec)
- Access
 - Select function in main display
 - context menu
 - Select function in flat profile
 - Code Display
 - Show Source Code



Xprofiler - Disassembler Code

address	no. ticks per instr.	instruction	assembler code	source code
10002E18	81	FC04287C	fmas 6, 4, 1, 5	
10002E1C	64	CCF70008	lfd 7, 0x8(23)	POLD(I, J) = P(I, J)+ALPHA*(PDEM(I, J)-
10002E20	187	C9C00008	lfa 8, 0x8(12)	UOLD(I, J) = U(I, J)+ALPHA*(USEM(I, J)-
10002E24	53	C9750008	lfa 11, 0x8(21)	
10002E28	89	FB63582A	fa 11, 3, 11	
10002E2C	63	FB28387C	fmas 9, 8, 1, 7	POLD(I, J) = P(I, J)+ALPHA*(PDEM(I, J)-
10002E30	4	D05B0008	stfd 10, 0x8(27)	U(I, J) = USEM(I, J)
10002E34		C9540008	lfd 10, 0x8(20)	VOLD(I, J) = V(I, J)+ALPHA*(VDEM(I, J)-
10002E38	113	FC0A802A	fa 6, 10, 6	
10002E3C	27	C8760008	lfa 3, 0x8(22)	POLD(I, J) = P(I, J)+ALPHA*(PDEM(I, J)-
10002E40	87	FB8012FA	fma 12, 0, 11, 2	UOLD(I, J) = U(I, J)+ALPHA*(USEM(I, J)-
10002E44	35	DCB90008	stfd 5, 0x8(25)	V(I, J) = VDEM(I, J)
10002E48	4	FC0A802A	fa 3, 3, 9	POLD(I, J) = P(I, J)+ALPHA*(PDEM(I, J)-
10002E4C	12	CB5A0008	lfd 10, 0x8(26)	UOLD(I, J) = U(I, J)+ALPHA*(USEM(I, J)-
10002E50	62	FC0211BA	fma 6, 0, 6, 4	VOLD(I, J) = V(I, J)+ALPHA*(VDEM(I, J)-
10002E54	36	C85B0008	lfa 2, 0x8(27)	UOLD(I, J) = U(I, J)+ALPHA*(USEM(I, J)-
10002E58	244	DC0C0008	stfd 7, 0x8(12)	P(I, J) = PDEM(I, J)
10002E5C	28	FB0040FA	fma 8, 0, 3, 8	POLD(I, J) = P(I, J)+ALPHA*(PDEM(I, J)-
10002E60		C8990008	lfd 4, 0x8(25)	VOLD(I, J) = V(I, J)+ALPHA*(VDEM(I, J)-
10002E64	316	DCB40008	stfd 6, 0x8(20)	
10002E68	29	FC62907C	fmas 3, 2, 1, 10	UOLD(I, J) = U(I, J)+ALPHA*(USEM(I, J)-

HPM: What Are Performance Counters

- Extra logic inserted in the processor to count specific events
- Updated at every cycle
- Strengths:
 - Non-intrusive
 - Accurate
 - Low overhead
- Weaknesses:
 - Specific for each processor
 - Access is not well documented
 - Lack of standard and documentation on what is counted

HPM: Hardware Counters Examples

- Cycles
- Instructions
- Floating point instructions
- Integer instructions
- Load/stores
- Cache misses
- TLB misses
- Branch taken / not taken
- Branch mispredictions
- Useful derived metrics
 - IPC - instructions per cycle
 - Float point rate (Mflip/s)
 - Computation intensity
 - Instructions per load/store
 - Load/stores per cache miss
 - Cache hit rate
 - Stores per load miss
 - Loads per store miss
 - Loads per TLB miss
 - Branches mispredicted %

LIBHPM

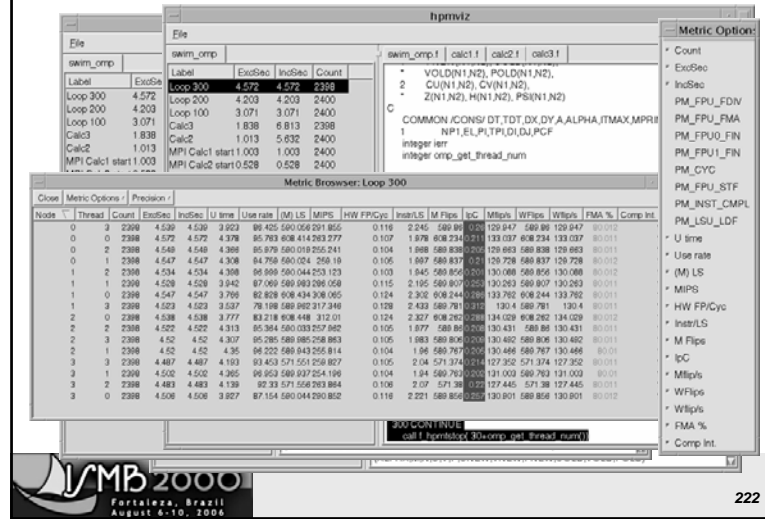
Go in the source code and instrument different sections independently

- Declaration:
 - #include f_hpm.h
- Use:
 - call **f_hpmunit(0, "prog"**)
 - call **f_hpmstart(1, "work"**)
 - do
 - call do_work()
 - call **f_hpmstart(22, "more work"**)
 - call compute_meaning_of_life()
 - call **f_hpmstop(22)**
 - end do
 - call **f_hpmstop(1)**
 - call **f_hpmterminate(0)**
- Supports MPI (OpenMP, threads on other PowerPC platforms)
- Multiple instrumentation points
- Nested sections
- Supports Fortran, C, C++



221

HPM Data Visualization



222

Blue Gene Check Systems

Sanity.rts

stdout[20]: MPI: 20/32, Pers: <0,1,1,0>/<4,4,2,1>, Torus? X0Y0Z0, VN? 0, Mem: 512MB(6), Loc: R00-M1-Nf-C:J14-U11

stdout[20]: MPI: 20/64, Pers: <0,1,1,0>/<4,4,2,2>, Torus? X0Y0Z0, VN? 1, Mem: 512MB(6), Loc: R00-M1-N2-C:J14-U11



223

Blue Gene PI – Monte Carlo

```

stdout[0]: #cpus #trials pi(est) err(est) err(abs) time(s) Mtrials/s
stdout[0]: 32 256000000 3.14176 0.00022 0.00017 1.082 236.58
stdout[0]: 16 256000000 3.14164 0.00022 0.00004 2.164 118.29
stdout[0]: 8 256000000 3.14157 0.00022 0.00002 4.328 59.15
stdout[0]: 4 256000000 3.14160 0.00022 0.00000 8.656 29.57
stdout[0]: 2 256000000 3.14155 0.00022 0.00004 17.313 14.79
stdout[0]: 1 256000000 3.14145 0.00022 0.00014 34.625 7.39
  
```



224

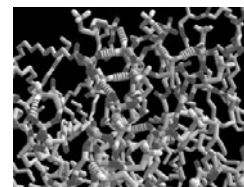


Application Performance

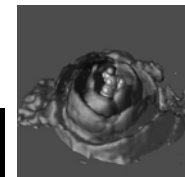
Brief overview of some applications

BGW at IBM T.J. Watson Research Center

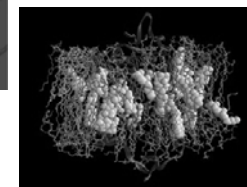
IBM has a team of life sciences researchers developing Blue Matter – application software used to run simulations of protein dynamics on Blue Gene. They are now running production science experiments on membrane proteins. Experiments that were taking a month or more on a conventional system are now taking a few days on Blue Gene.



Lipids critical to cell division and fusion



Omega-3 Fatty acids and cholesterol



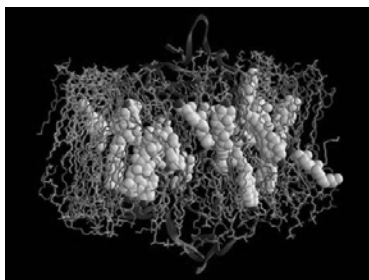
G Protein-Coupled Receptors (GPCR) in a membrane environment



Blue Matter: RHODOPSIN GPCR

Diseases associated with malfunction of GPCRs are:

- Congestive Heart Failure
- Hypertension & Stroke
- Cancer
- Ulcers
- Allergies
- Asthma
- Anxiety
- Psychosis
- Migraines
- Parkinson's Disease

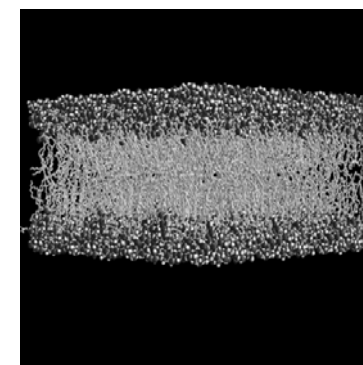


Membrane Proteins
Cell Signaling, Ion/Nutrient Transport, Targets of Many Drugs

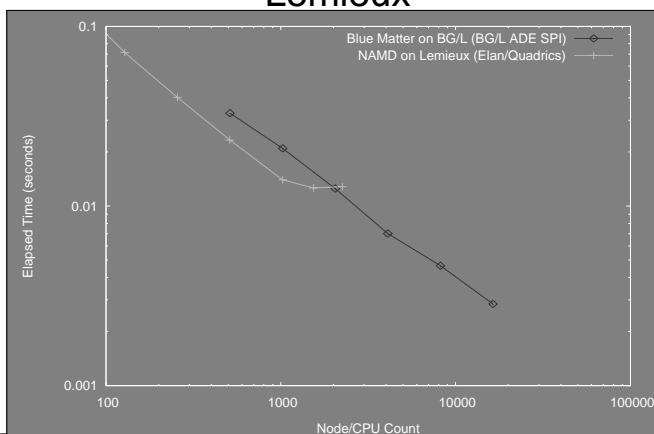


Blue Matter: Lipid Bilayers

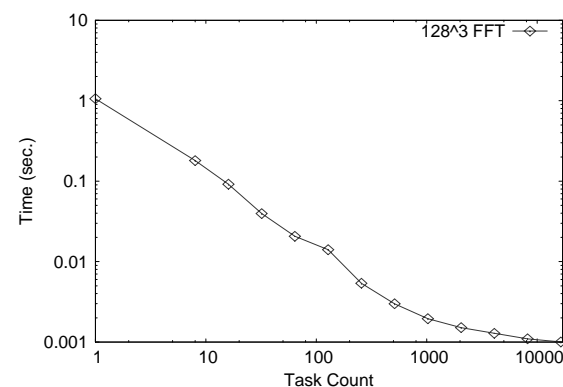
- Lipids provide the environment for membrane proteins and enable critical functions including cell signalling and cell division.
- Studying lipids is crucial to understanding diseases related to these proteins, including muscular dystrophy and Alzheimer's.
- One third of all proteins in the human body -- and half of all drug targets -- are membrane proteins



Blue Matter on BG/L vs. NAMD on PSC Lemieux



3D-Fast Fourier Transform



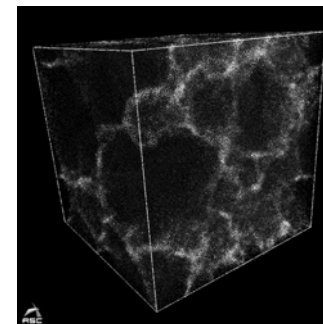
Enabled by optimized MPI Alltoall[v]

Applications Performance on Blue Gene

- BG is first HPC system to break barrier of 100+ TeraFlop/s sustained performance on real applications (Molecular Dynamics)
 - ddcMD – 101.5 TeraFlop/s (7 hrs of Uranium atoms on 64 racks)
 - CPMD – 110.4 TeraFlop/s
- Several other applications have achieved two orders of magnitude or more higher performance than previously possible – successful scaling achieved from 1K to 100K processors
- Gordon Bell Prize competition at SC 2005
 - 4 of 6 finalists based on Blue Gene
 - LLNL/IBM team won for “100+ TFlop Solidification Simulations on Blue Gene/L”
 - AIST also captured Best Technical Paper

ddcMD - Classical MD 2005 Gordon Bell Prize Winner

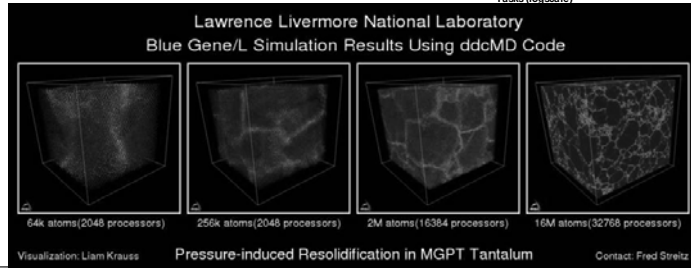
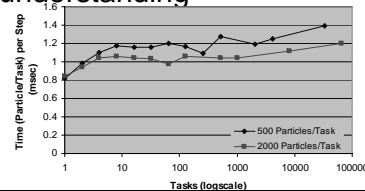
- Scalable, general purpose code for performing classical molecular dynamics (MD) simulations using highly accurate MGPT potentials
- MGPT semi-empirical potentials, based on a rigorous expansion of many body terms in the total energy, are needed in to quantitatively investigate dynamic behavior of transition metals and actinides
- Visualization of important scientific findings already achieved on BG/L: Molten Ta at 5000K demonstrates solidification during isothermal compression to 250 GPa



524 million atom simulations on 64K nodes are orders of magnitude larger than any previously attempted runs; *superb strong and weak scaling* expected for full machine - (“very impressive machine” says PI)

Excellent scaling of ddcMD on BG/L supports solidification understanding

- Nucleation is initiated at multiple independent sites in each sample cell
- Growth of solid grains initiates independently, but soon leads to grain boundaries which span the simulation cell
- 101.5 TF on 64 racks
- The ddcMD team is currently using 131,072 CPUs of BG/L for unprecedented
- five hundred million atom MGPT simulations



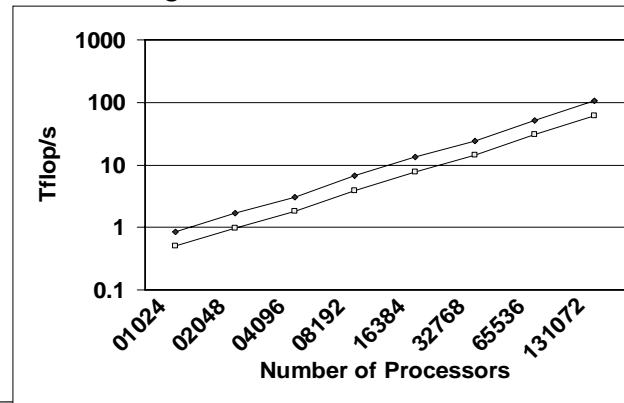
Lawrence Livermore National Laboratory
Blue Gene/L Simulation Results Using ddcMD Code

Visualization: Liam Krauss Pressure-induced Resolidification in MGPT Tantalum

Contact: Fred Seitz

233

Performance of ddcMD on Blue Gene Weak scaling: MGPT Uranium and Tantalum

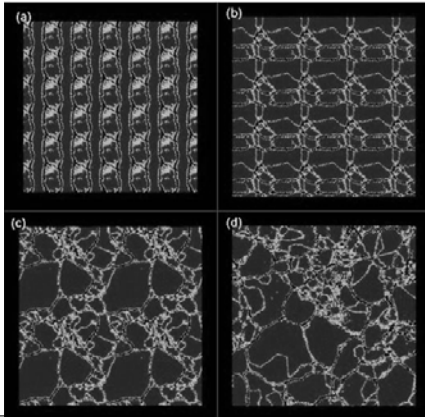


Lawrence Livermore National Laboratory
Blue Gene/L Simulation Results Using ddcMD Code

234

ddcMD Simulation Results

(a) 64K atoms, (b) 256 K atoms, (c) 2,048,000 atoms, (d) 16,384,000 atoms



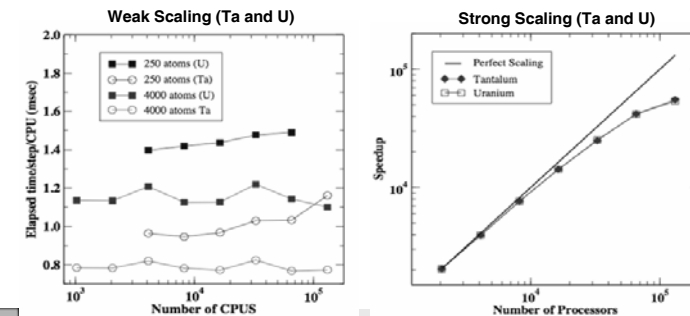
Lawrence Livermore National Laboratory
Blue Gene/L Simulation Results Using ddcMD Code

235

Scaling ddcMD up to 131,072 CPUs

... but allows unprecedented scaling of size or time

- Weak scaling is virtually flat across the entire machine - enables simulation of tens of billions of atoms (roughly a cubic micron of material)
- Strong scaling shows speedup down to 8 atoms/CPU - enables simulations involving millions of steps (typically ns of simulated time)



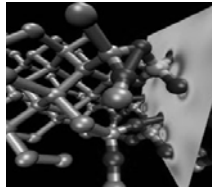
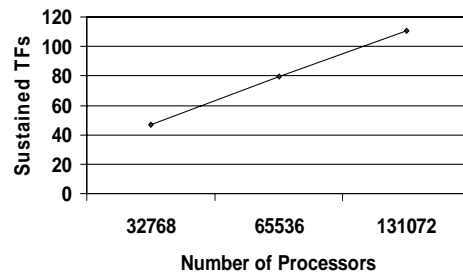
Lawrence Livermore National Laboratory
Blue Gene/L Simulation Results Using ddcMD Code

236

CPMD

Alessandro Curioni, Salomon Billeter, Wanda Andreoni

CPMD Performance on BG/L



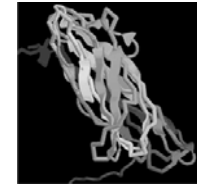
Developed at IBM Zurich from Car Parinello code
 Uses Plane Wave Basis functions, FFT, MPI Collectives
 Ongoing project : IBM/LLNL Pd:H (~900 atoms) Hydrogen Storage
 Achieved 110.4 Teraflop/s sustained on 64 racks BG/L (excellent strong scaling)



237

AIST

One of our biggest research challenges is to apply data obtained from genome decoding to protein engineering and drug design. The scale of simulation this requires cannot be done without the help of supercomputers. IBM's Blue Gene/L supercomputer provides us with a massive supercomputing resource that will dramatically accelerate our work.



Dr. Yutaka Akiyama, Director, Computational Biology Research Center, National Institute of Advanced Industrial Science and Technology (AIST)



National Institute of
Advanced Industrial Science and Technology
独立行政法人 産業技術総合研究所

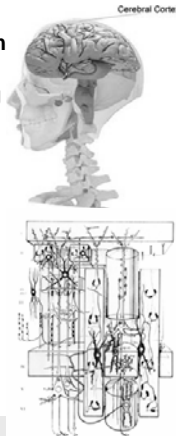


238

EPFL

IBM's Blue Gene supercomputer allows a quantum leap in the level of detail at which the brain can be modeled. The time has come to begin assimilating the wealth of data that has accumulated over the past century and begin building biologically accurate models of the brain to aid our understanding of brain function and dysfunction.

*Henry Markram, Laboratory of Neural Microcircuitry, Brain Mind Institute
 Ecole Polytechnique Fédérale de Lausanne
 Switzerland*



239

14th Annual International Conference On Intelligent Systems For Molecular Biology



Massively Parallel Computing Environment

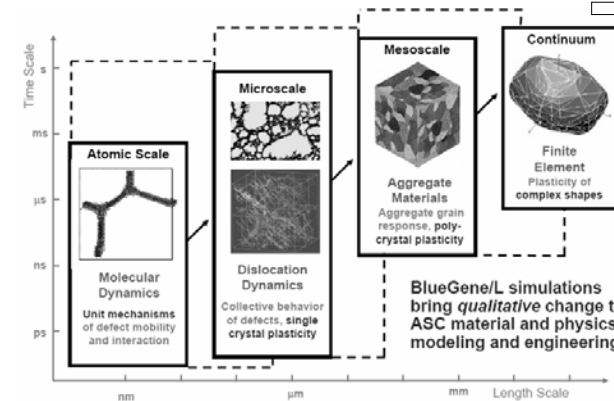
What might we do?

ISMBS 2006 - Fortaleza, Brazil - August 6-10, 2006

The Real Question

- What can you do with 130K processors? (8K, 16K, 32K)
 - Really BIG problems – Maybe
 - Same problems but much finer resolution, refinements, larger searches in shorter time – Maybe
 - Explore parameters – large parameter space – Maybe
- BUT
 - Perhaps need to rethink the problem
 - Most parallel programs are Single Program Multiple Data
- What if
 - Multiple Programs Multiple Data - - Systems of Complex Systems interacting?
 - Handle multi-scale, multi-physics - - Biology is multi-scale?

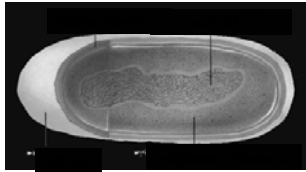
BlueGene/L will allow overlapping evaluation of models for the first time



Grand Challenges for Healthcare & Life Sciences

• Whole cell modeling - e.coli

Genetically engineer e-coli strains to increase drug production by fermentation



- E. coli is the most popular target
 - It's simple: "only" ~4000 genes, no nucleus, unitary genes, no organelles
 - It's well studied
- How might Blue Gene impact
 - Atomistic level
 - Chemical kinetics
 - Continuous models

• Drug delivery modeling

Develop manufacturing processes that insure the right dose of a drug is effectively delivered



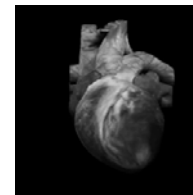
- Inhaler delivery of drugs
 - Multi-scale – atomistic, chemical, fluid flow
 - Manufacturing complex model
- How might Blue Gene impact
 - Atomistic level
 - Chemical kinetics
 - Continuous models

Spatially explicit model of actin and myosin interaction in the cardiac myofilament - Possible simulation mapping on BG/L- Jeremy Rice, Jagir R Hussan, Pieter P. de Tombe, Gustavo Stolovitzky, Yuhai Tu

Mathematical modeling of heart will allow better therapies for heart disease..

...but modeling heart requires bridging between organ level and molecular level

Organ level



Reconstruction of whole heart by Peter Hunter, U. of Auckland

Cell level



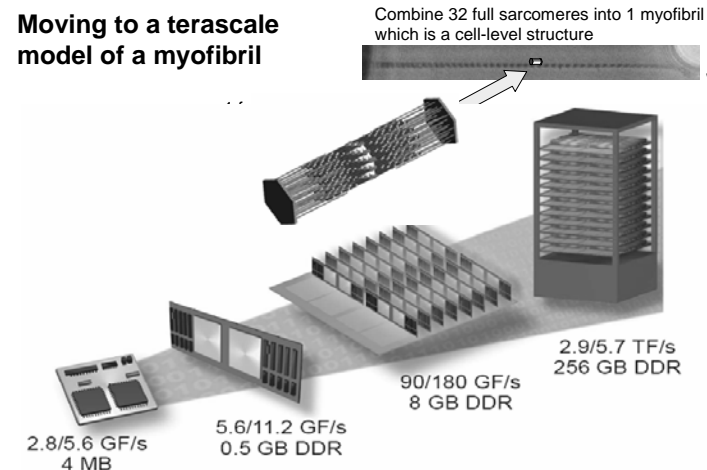
In each cell of heart, a lattice of sarcomeres produce contraction on every heart beat.

Molecular level



Sarcomere contracts by cyclical interactions of myosin on thick filament (red) and actin in thin filament (green).

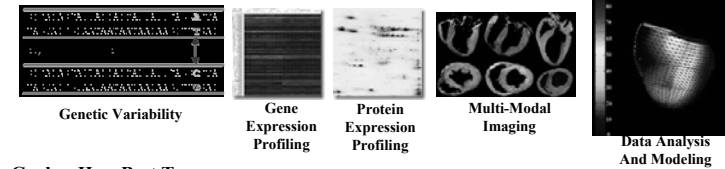
Moving to a terascale model of a myofibril



245

Increasing Importance of Engineering, Mathematical and Computational Sciences in Human Disease Research – Computational Medicine

Multi-Scale Patient-Specific Data



Goals – How Best To:

- analyze these data sets to gain novel insights regarding disease mechanisms and to perform risk prediction targeted to the individual (*statistical inference, pattern discovery/classification, computational anatomy*)
- synthesize computational models of biological systems and disease processes that provide insights into disease mechanisms and novel therapies (*dynamical systems theory, probability theory, stochastic processes*)
- Distribute multi-scale data, data analysis methods and computational models to basic and clinical researchers through computational grids (“bio-grids”) – Blue Gene potential compute engine

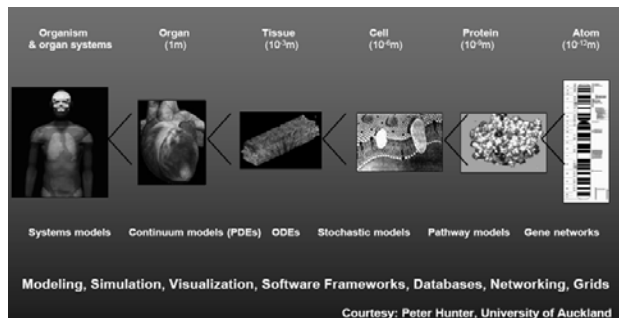


Courtesy: Rai Winslow, Johns Hopkins University

246

Multi-scale in Physiome Project

Hunter's Group – converting CMISS (Continuum Mechanics, Image analysis, Signal processing and System Identification) to potentially use Blue Gene



- Blue Gene Impact
 - Atomistic to System Model scales – tightly coupled



247

Biomedical and Molecular Imaging – Exploit Parallelism

High Resolution Research Tomograph (HRRT)

Resolution ~ 2.5 mm
Sensitivity ~ 6%
Number of detectors 119,800
Number of Lines of response (LOR) 4.5 Billion !!

Dynamic (4D) PET Imaging

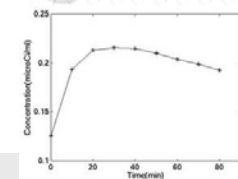
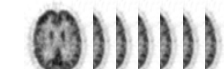
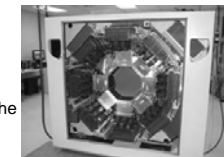
Quantity measured in PET: *in vivo* regional concentration of the radiotracer

- Use multiple time frames to “measure” the physiologic or metabolic process
- Can extract how various compartments interact

Radiology - Storage/Computational Issues

- ~30GB of raw/list-mode data per study
- Each study divided into ~30 frames and reconstructed
- Currently, computation takes:
 - 15 hours (span3), 7 hours (span9:lower-res) per study!
 - 8 nodes/frame, 4 frames processed at a time
 - Each reconstructed image is 50MB (i.e. 1.5GB /study)

Goal: 20 studies/week: clearly not achieved in span3



248

Acknowledgements

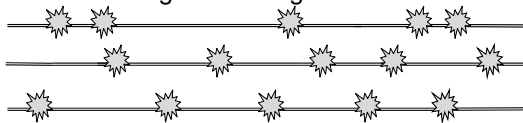
- Heart Models
 - John Jeremy Rice, Jagir R Hussan, Gustavo Stolovitzky, Yuhai Tu – IBM and Pieter P. de Tombe - University of Illinois Chicago
 - Rai Winslow – Johns Hopkins
 - Peter Hunter – U. of Auckland
- Multi-scale
 - Steve Louis - LLNL
- PET Imaging
 - Dean Wong, Arman Rahmim – Johns Hopkins
- Transcription Factors
 - Charles DeLisi, Boris Shakhnovich, Tim Reddy, J. Max Harvey – Boston University

In-Depth look at Large-Scale Computational Genomics on the IBM Blue Gene/L Supercomputer

Genome Assembly

Input: Multiple copies of the same genome

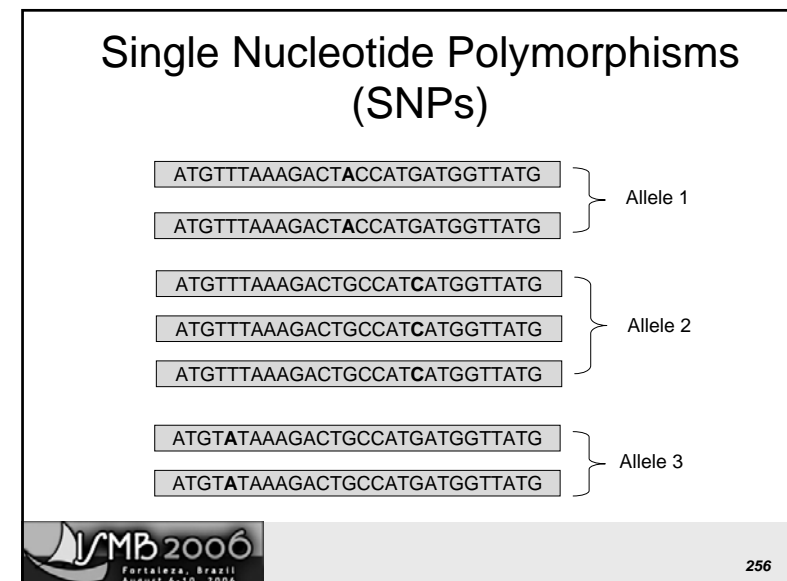
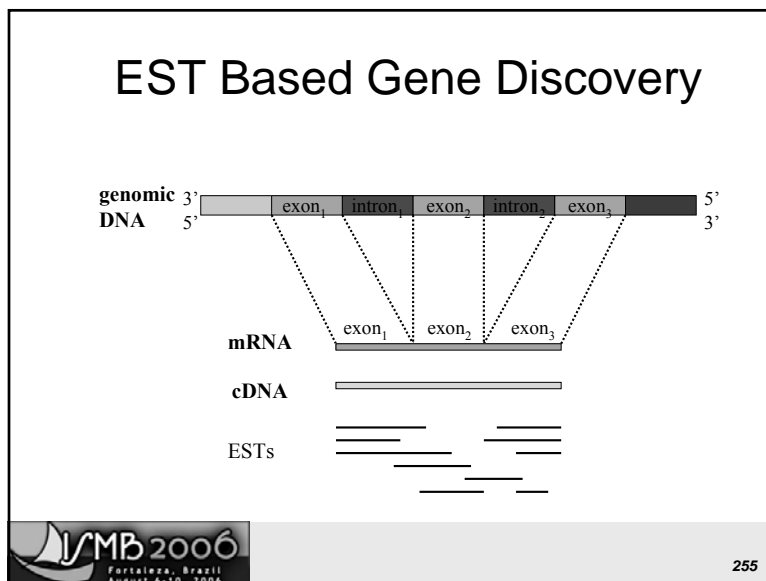
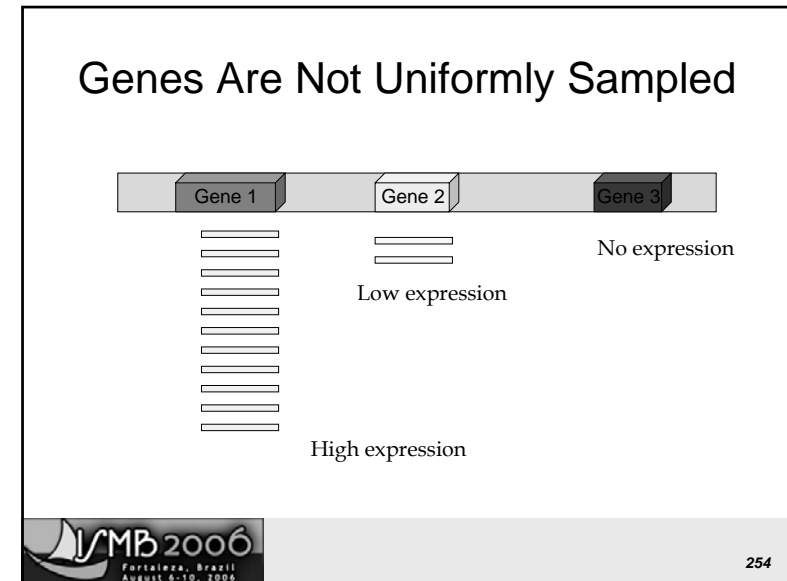
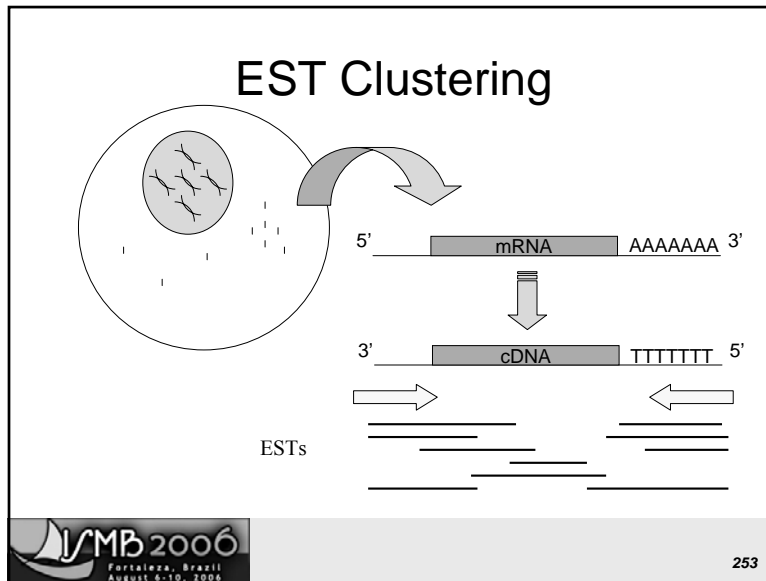
Output: Unordered genome fragments



Process: Randomly fragment each copy

Sequence Assembly Required!





SNPs Based on Assembly

ATGTTTAAAGACTACCATGATGGTTATG	
ATGTTTAAAGACTACCATGATGGTTATG	
ATGTTTAAAGACTGCCATCATGGTTATG	
ATGTTTAAAGACTGCCATCATGGTTATG	
ATGTTTAAAGACTGCCATGATGGTTATG	
ATGTTTAAAGACTGCCATGATGGTTATG	
ATGT?TAAAGACT?CCAT?ATGGTTATG	Consensus

Alignment of related genomic sequences

SNPs Based On Clustering

ATGTTTAAAGACTGCCATGATGGTTATG	Genome
ATGTTTAAAGACTACCATGATGGTTATG	
ATGTTTAAAGACTACCATGATGGTTATG	
ATGTTTAAAGACTGCCATCATGGTTATG	
ATGTTTAAAGACTGCCATCATGGTTATG	
ATGTTTAAAGACTGCCATCATGGTTATG	
ATGTTTAAAGACTGCCATGATGGTTATG	
ATGTTTAAAGACTGCCATGATGGTTATG	

Samples that are aligned to the consensus

Naïve Approach

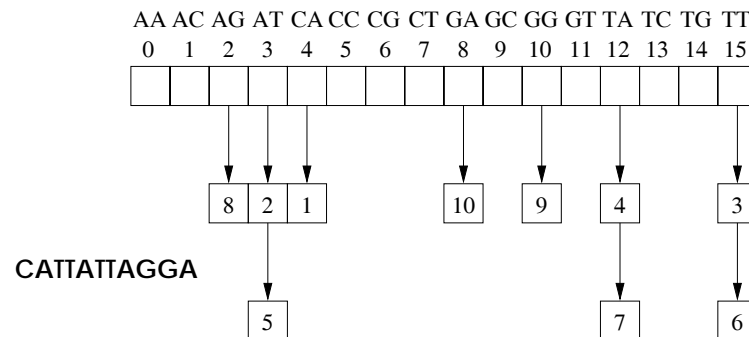
All vs. All alignments + post processing
 Compute-intensive and wasteful!

- 33 million fragments for mouse assembly
- 7+ million human ESTs

Typical Methodology

- Identify pairs of fragments that have a good exact match (promising pairs).
- Restrict alignment computations to promising pairs.
- Perform post-processing.

Lookup Table Pair Generation



Problems for Large-scale Analysis

- Longer matches are revealed as multiple short matches.
- Matches are arbitrarily generated.
- Linear space for uniformly random overlaps with constant coverage but worst-case quadratic in the non-uniform case.

PaCE Methodology

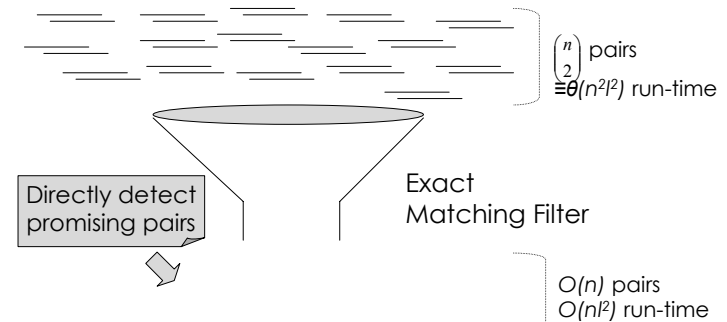
- Reduce space requirement from quadratic to linear.
- Generate promising pairs in decreasing order of maximal common substring length.
- Constant time per generation of a pairwise maximal common substring.
- Significantly reduce number of alignments without affecting quality.
- Massively parallel processing – reduce run-time; increase available memory.

A Specific Application: Maize Genome Assembly

Why sequence the maize genome?

- Maize (*i.e.*, corn) is an economically important crop.
- Best studied model organism for the cereal crops.
- Just as the human genome project will intensify upcoming medical advances, cereal genomes (rice and maize) will help improve worldwide food production.

Typical Assembly Strategy



Genome Assembly Example

- Human Genome Assembly (Venter *et al.* 2001):
 - Input: 27 million fragments
 - Program: Celera Assembler
 - 10,000 CPU hours for detecting overlaps
 - Parallelized to run on 64 GB shared memory machine + 10 4-processor SMPs with 4-GB memory
 - 10,000 CPU hours for the rest

Maize Genome Assembly

- Maize genome is comparable in size to the human genome (2.5 GB) but is highly repetitive (65-80%). About 15-20% is gene space.
- NSF Workshop in July 2001 to debate sequencing strategies

Maize Genome Assembly

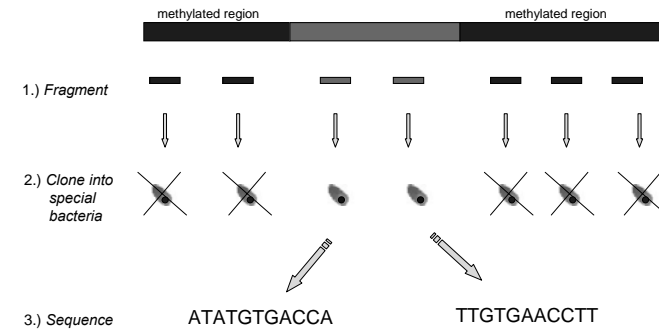
NSF funded pilot projects (2002; \$10.2 million):

- “gene-enrichment” – Consortium for Maize Genomics (Danforth Center, TIGR, Purdue & Orion Genomics)
 - Methylation filtration (MF)
 - High C_0t selection (HC)
- BAC sequencing – Rutgers & Univ. of Arizona.
- Dept. of Energy (DOE) added about 2.4 million fragments.



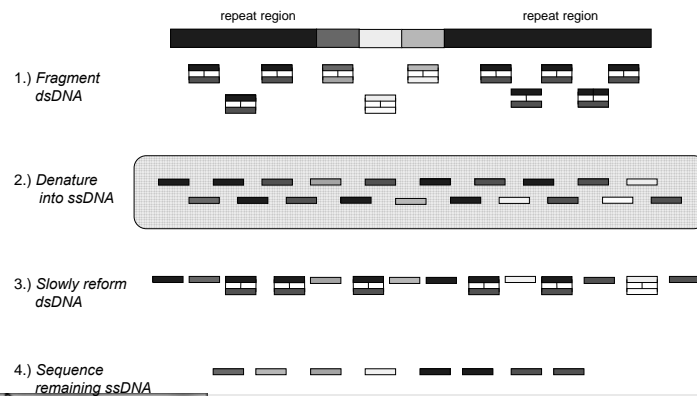
269

Methylation Filtration



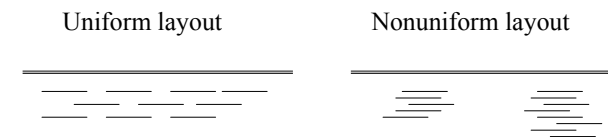
270

High C_0t Selection



271

Random vs. Biased Sampling



- Uniform case – $O(n)$ overlaps
- Non-uniform case – $O(n^2)$ overlaps



272

PaCE Methodology

- First cluster, then assemble.
- Two sequences fall in the same cluster if there is a chain of overlaps that leads from one sequence to the other.
- Each cluster can be assembled into a contig.



273

Clustering Strategy

- Initially, treat each sequence as a cluster by itself.
- If two sequences from two different clusters show significant overlap, merge the clusters.
- Use union-find data structure.



274

Processing High-quality Overlaps first is important!

Successful overlap results in

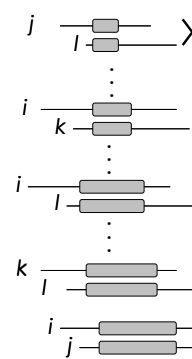
- Merging of two clusters.
- No need to test other promising pairs of fragments where a member of the pair comes from each constituent cluster.



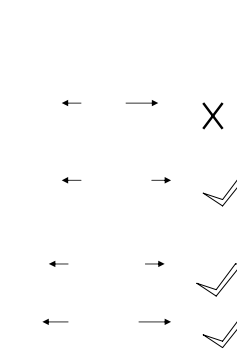
275

Clustering Heuristic

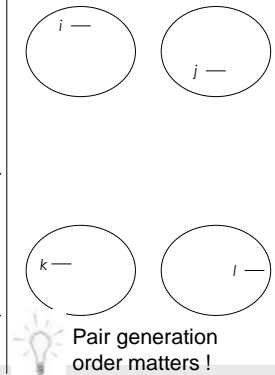
Promising pairs:



Pairs aligned:



Clustering:



Pair generation
order matters !

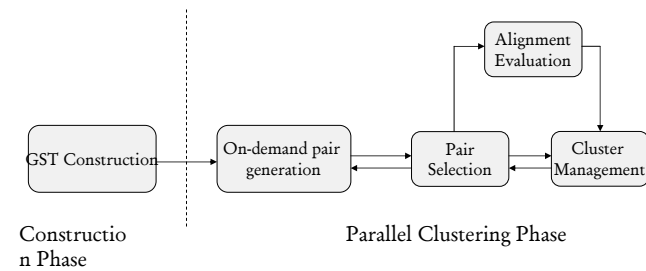


276

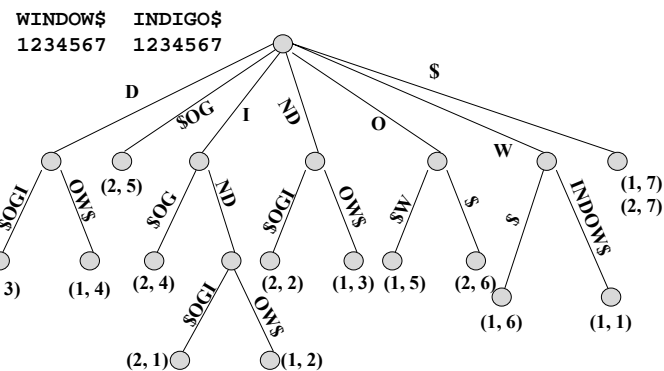
Pair Generation Methodology

- Generate pairs
 - In non-increasing order of maximal common substring length
 - On-demand without storing previously generated pairs
 - $O(1)$ amortized time per pair
 - Using linear space

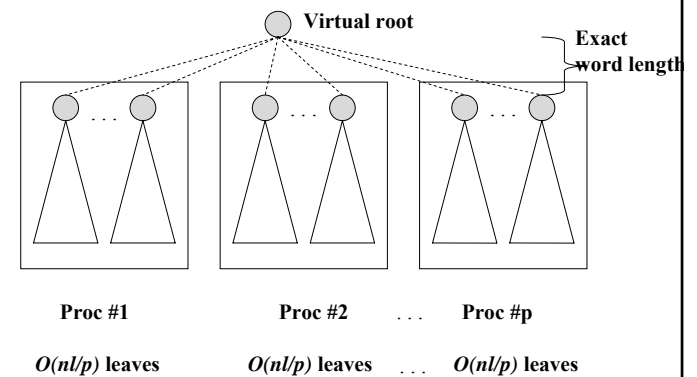
PaCE Software Architecture



Generalized Suffix Tree (GST)



Parallel Construction of GST



Parallel Construction of GST

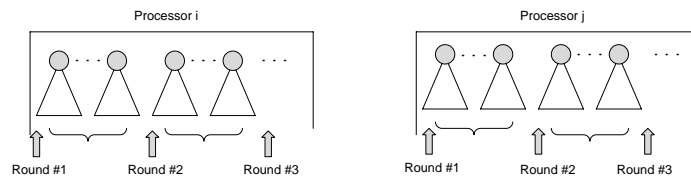
- Bucket the suffixes of the sequences based on the first k bases.
- Redistribute the suffixes in parallel such that each processor owns a set of buckets.
- Build GST locally in each processor.
- In each processor, $\#leaves = O(nl/p)$
- Run-time = $O(nl^2/p)$

GST Construction: Scaling Issues

- How to acquire sequences corresponding to the suffixes contained in the local buckets ?
- **Approach (a):** Acquire sequences from disk before constructing each subtree
- **Issues:** (i) Requires random access I/O in parallel, and (ii) the same sequence can be read multiple times for different buckets

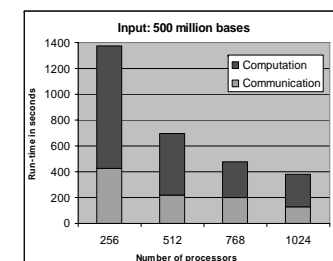
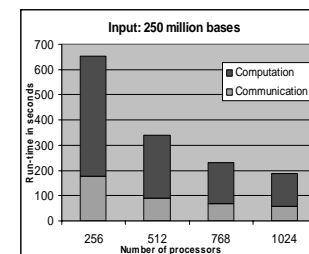
BG/L-Specific Optimizations

- **Approach (b):** Process buckets in batches and acquire sequences by communicating before every batch construction



- Each communication round is an Alltoallv
- Number of rounds and data communicated half for every doubling of processors

GST Construction on BlueGene/L

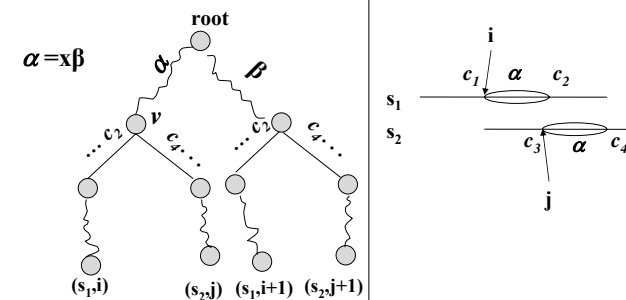


Pair Generation Algorithm

- Process the nodes in the local GST in the decreasing order of string-depth and generate pairs at each node.
- Generate a pair at a node only if the corresponding overlap is maximal.

Main Idea of the Algorithm

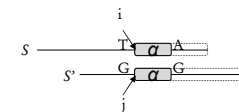
- *Maximal common substring*



Left Character Sets (*lsets*)

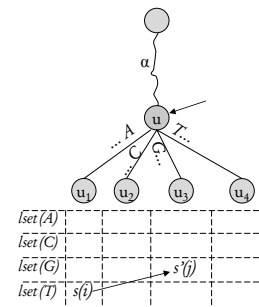
- *leaf-set*(v) = set of strings whose suffixes are present in the subtree of v .
- *lset* (v) = partition of *leaf-set*(v) into $|\Sigma|+1$ subsets, $l_A(v)$, $l_C(v)$, $l_G(v)$, $l_T(v)$, $l_\lambda(v)$.

Maximal Match Detection



- **Right Maximality**
 $\equiv s(i)$ and $s'(j)$ are in subtrees of two different children of u
- **Left Maximality**
 $\equiv s[i-1] \neq s'[j-1]$, if $i > 1$ and $j > 1$

Pair generation at an internal node u

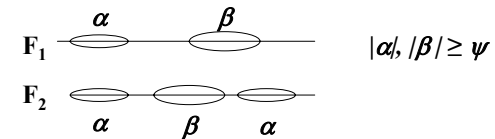


Run-time: $O(1)$ per pair

Run-time for Pair Generation

- Sorting of nodes in the local GST
 $= O(nl/p)$
- Processing of all nodes in the local GST
 $= O(\text{\# pairs generated})$

Number of Duplicates

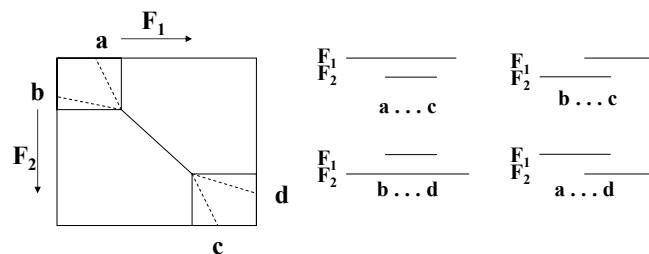


eg., (F_1, F_2) is generated at most twice.

of times a pair is generated

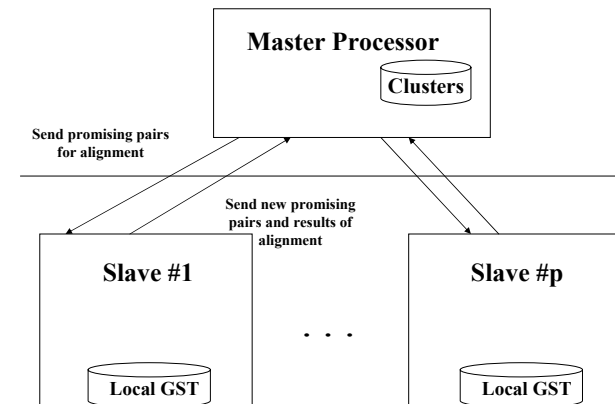
\leq # of distinct maximal common substrings
 (of length $\geq \psi$)

Possible Fragment Overlaps

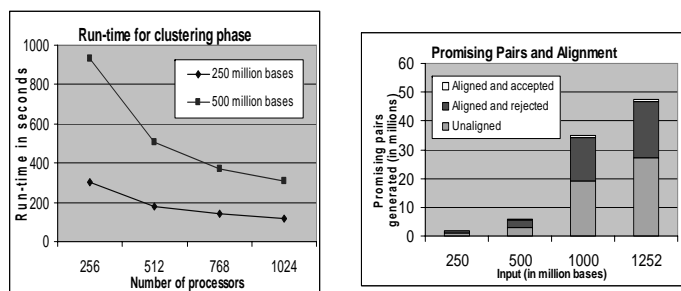


- Compute only lower and upper rectangles
- Do banded dynamic programming

Parallel Clustering Phase

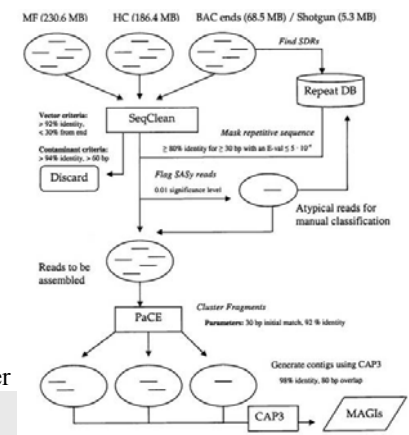


Clustering Phase Performance on BlueGene/L



Overview of Assembly Pipeline

- 1.) Collect data
- 2.) Clean up data
- 3.) Mask repeats
- 4.) Cluster data
- 5.) Assemble smaller



Maize Assembly on BlueGene/L



Number of Input Bases (in billions)	Number of nodes	PaCE Runtimes (in minutes)		
		Tree Construction	Clustering	Total
1.25	1,024	13	89	102

Maize Assembly on BlueGene/L

Number of Input Bases (in billions)	Number of processors	PaCE Runtimes (in minutes)		
		Tree Construction	Clustering	Total
0.5	8,192	1.2	11	13
1.15	8,192	2.3	72	75

Maize Assembled Genomic Islands (MAGIs)

	MAGI v4.0
<i>Input Sequences</i>	3,202,268
<i>Assembly Size</i>	329.61 MB
<i>GC Content</i>	44.9%
<i>Contigs</i>	217,106
<i>Non-repetitive Singletons</i>	567,797
<i>Avg contig len</i>	1,518
<i>Avg GSS per contig</i>	4.78

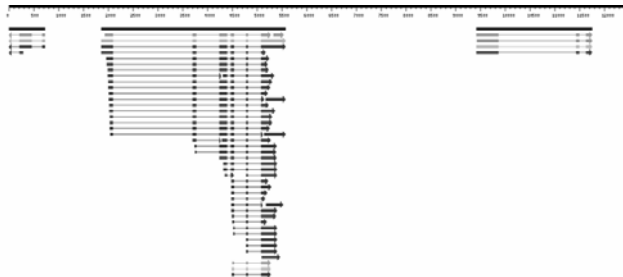


MAGI 3.1 quality and coverage

Gene	Length	Error Rate	Errors	Alignment Length
<i>gl8a</i>	6760	0.0	0	1512
<i>pd2</i>	5443	0.0	0	3462
<i>pd3</i>	7773	3.5e-4	1	2793
<i>rf2a</i>	11 520	0.0	0	1886
<i>rf2b</i>	4311	3e-4	1	3315
<i>rf2c</i>	7257	0.0	0	3898
<i>rf2d</i>	7415	0.0	0	3880
<i>rf2e</i>	4739	0.0	0	1765
<i>rth1</i>	14 350	4.5e-4	1	2190
<i>rth3</i>	3152	0.0	0	3145
<i>Sum</i>	72 720	1e-4	3	27 846

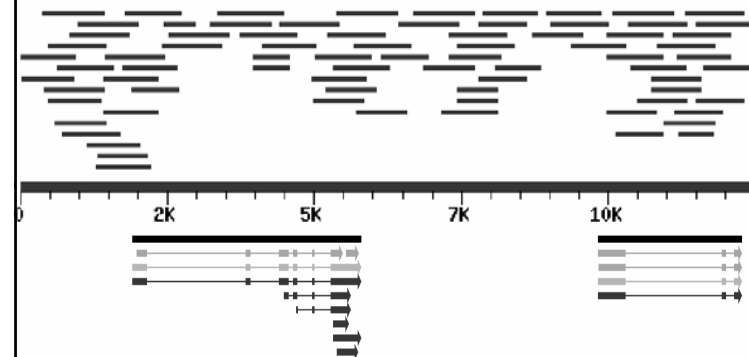
Gene “archipelagoes”

MAGI3.1_4593 (12,498 bp)



Gene “archipelagoes”

MAGI_4593



Maize assembly Portal

MAGI Maize Assembled Genomic Island

WELCOME TO THE MAGI WEBSITE, WHICH REPORTS THE RESULTS OF A MAIZE GENOME ASSEMBLY PROJECT BEING CONDUCTED BY THE ALURU, ASHLOCK AND SCHNABLE RESEARCH GROUPS.

As the best-studied biological model for cereals and one of the world's most important crops, there is a strong rationale for sequencing the maize genome (Bennetzen et al., 2001; Chandler and Brendel, 2002) and the National Science Foundation has recently announced an [EET](#) to do so. Pilot studies have already generated substantial numbers of gene-enriched genomic survey sequences (GSSs; Whitelaw et al., 2003; Palmer et al., 2003), as well as BAC sequences and random shotgun GSSs from maize and sorghum that are available for download from [Maizegenome.org](#).

We have recently reported the development of innovative parallel algorithms for the efficient assembly of non-uniformly sampled genomic fragments (such as gene-enriched GSSs) into "genomic islands" (Emrich et al., 2004). We have used these procedures and a 64 processor IBM xSeries cluster to assemble ~850,000 maize GSSs generated by the Consortium for Maize Genomics into MAGIs (Maize Assembled Genomic Islands). We have similarly assembled ~500,000 gene-enriched sorghum (ATx623) GSSs generated by [Cotton Genomics](#) and their partners [NC-Hybrids](#) and [Solvigen](#) into SAMIs (Sorghum Assembled Genomic Islands).

Based on computational and biological quality assessments it appears that a very high percentage of genic MAGIs and SAMIs accurately reflect the structure of the maize and sorghum genomes (Fu et al., submitted).

To identify genomic contigs associated with particular genes or functions, MAGIs and SAMIs may be searched using BLAST. In addition, MAGIs have been annotated via sequence similarity, alignments to ESTs using [GeneSinger](#) and the *ab-initio* gene prediction tool [ESGENE](#) (Zhu et al., submitted; Fu et al., submitted). The GSSs that comprise each MAGI can also be displayed. It is also now possible to request that specific MAGIs be

Sorghum Assembled genomic Islands (SAMIs)

	SAMI v1.0
Input Sequences	511,512
Assembly Size	98.46 MB
GC Content	45.12%
Contigs	74,673
Singletons	131,610
Avg contig len	1,319
Avg GSS per contig	5.08



302

More Information ...

Publications:

- **On Maize Assembly**
 - *Bioinformatics*, January 2004.
 - *International Parallel and Distributed Processing Symposium*, April 2006.
- **On PaCE**
 - *IEEE Transactions on Parallel and Distributed Systems*, December 2003.
 - *Nucleic Acids Research*, March 2003.
- **On Maize Genomics**
 - *Proc. National Academy of Sciences*, August 2005.



303

More information ...

- **PaCE software download**
 - <http://www.ece.iastate.edu/~aluru/software/PaCE>
 - Over 45 academic/governmental/non-profit users from 10 countries.
 - 2 companies.
- **Maize Assembly Website**
 - <http://www.plantgenomics.iastate.edu/maize>
 - Used by researchers from Berkeley, Cornell, Purdue, Penn State, Dupont, BASF etc.



304

Future of Maize Genome Sequencing Project

- US \$32 million project by NSF, DOE, and USDA for large-scale sequencing.
- Goal is to sequence all genes, determine their order and orientation, and anchor them to genetic/ physical maps.
- Projects started November 15, 2005.



305

\$32 million B73 maize genome sequencing consortium

Washington University*

Iowa State University



University of Arizona

Cold Spring Harbor

Courtesy of the NSF



306



Another Application: Mouse EST Clustering

- Input:
 - A random subset of 56,470 UniGene clusters downloaded in March 2006
 - 3.78 million total entries including ESTs and full-length cDNAs
- Output:
 - 60,862 clusters with more than one sequence
 - Average cluster = 55; Largest = 807,671
 - 83% of clusters are composed of a single UniGene

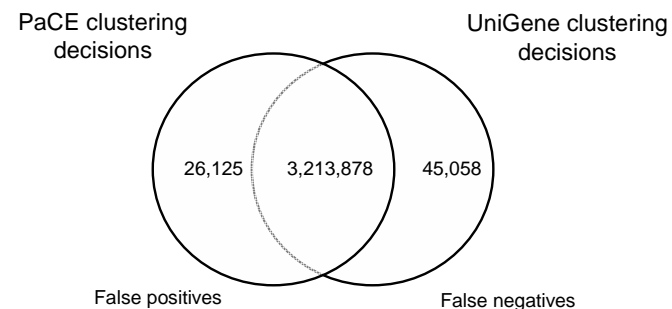


308

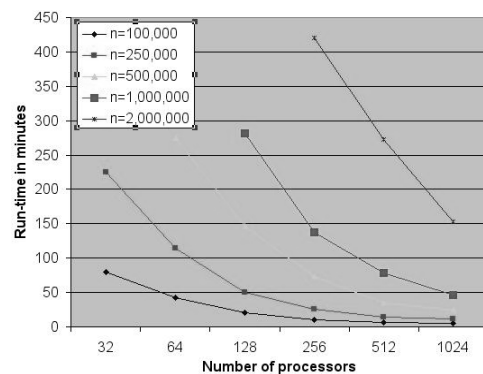
Validation

- Single-linkage clustering performs at most n merges.
- When comparing to UniGene, one measure of accuracy is the number of additional or missed merges performed.
- Ignoring clusters of size 1, our data suggest that over 98% of the links in UniGene were correctly determined by PaCE.

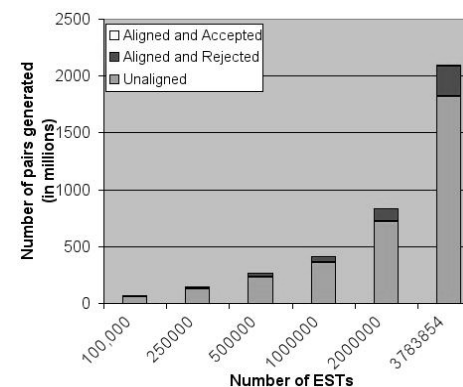
Clustering accuracy



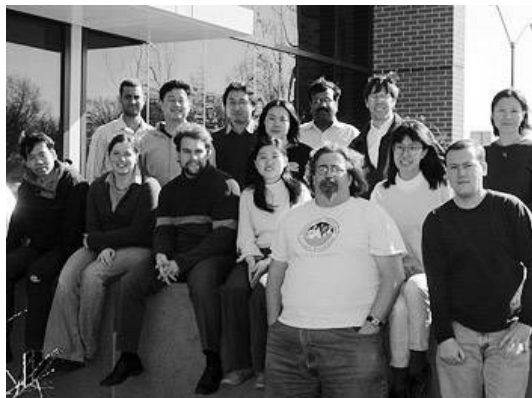
Run-time Scaling: Mouse EST Clustering



PaCE: Promising Pairs Statistics



Acknowledgements



Closing Remarks

Getting Started

<http://www.mcs.anl.gov/bgconsortium> & Activities



- Working Groups meetings:
 - Application Working Group
 - Help sponsor Porting Workshops – will continue
 - Technical meetings
 - Some access to ANL BG/L Machine
 - Systems Software Group
 - Help sponsor technical meetings/workshops
- Community building through web presence
 - BG Consortium Website
 - BG Consortium Wiki
 - BG Consortium email discussion
- Opportunities with IBM
 - BGW Consortium Days
 - Breakthrough science meeting

Remarks

- Is Blue Gene a system for computational Biology?
 - Starting to see effective use of lots of processors in this domain
- Still need to re-think how tackle problems - -
- Think of problems might tackle that until now would not dream
- Systems of complex systems will need multi-disciplinary teams - -
- The answers remains left to the audience/the reader/the users - - you!!

