Grid Computing: Application Development

Lennart Johnsson
Department of Computer Science and
the Texas Learning and Computation Center
University of Houston
Houston, TX
Department of Numerical Analysis and Computer Science and PDC
Royal Institute of Technology
Stockholm, Sweden
Grids are “Hot”

Courtesy Ken Kennedy
Grids: Application Development

“Cactus”
“Zeus-MP” “Tardis”
“NetSolve” “GTomo”
“SF-Express”
“Distributed Viz”

- How will my software behave on the projected hardware configuration? (performance)
- How will it behave dynamically? (robustness)
- How will it interact with other Grid applications an uses of the system?
- How can I make this a robust, stable, reusable application?

Courtesy Ken Kennedy
Grids: Middleware Development

- Libraries - network, performance instrumentation, runtime environment (e.g. Globus)
- Program Preparation System - dynamic compilers, runtime, etc.
- Do these things work and how well?
- With what applications and what range of applications?

Courtesy Ken Kennedy
Grids: Administration

- What if I change my resource access policies?
- What if I add/take away these resources?
- What if I change the “price” charged for resources?
- What happened to my Grid when it melted down last week?

Courtesy Ken Kennedy
Grid Application Development

- The GrADS vision: **federated** computers and software
  - An “application” is constructed dynamically from services and components on the network—selected to meet requirements
  - A “computer” is a dynamically constructed collection of processors, data sources, sensors, networks—optimized for our application

- And thus
  - Reduced barriers to access mean that we do much more computing, and more interesting computing, than today => Many more components (& services): massive parallelism
  - Distributed resource ownership => Sharing (for fun or profit) is fundamental; so are trust, policy, negotiation, payment
  - Computing is performed, increasingly, on unfamiliar systems => Dynamic behaviors, discovery, adaptivity, failure

- **Challenge:** exploring such future scenarios today, in compelling yet realistic settings
  - Identify, address **fundamental** issues (beyond RPC syntax of the day)
Grid Application Development: A Software Grand Challenge

- Goal: Reliable performance on heterogeneous platforms, under varying load on computation nodes and on communications links

- Challenges:
  - Presenting a high-level application development interface
    - If programming is hard, its useless
  - Designing and constructing applications for adaptability
  - Mapping applications to dynamically changing configurations
  - Determining when to interrupt execution and remap
    - Application monitors
    - Performance estimators
What is Needed

• Execution infrastructure for **reliable performance**
  - Automatic resource location and execution initiation
  - dynamic configuration to available resources

• Performance monitoring and control strategies
  - deep integration across compilers, tools, and runtime systems
  - performance contracts and dynamic reconfiguration

• Abstract Grid programming models and easy-to-use programming interfaces
  - design of an implementation strategy for those models
  - problem-solving environments

• Robust reliable numerical and data-structure libraries
  - predictability and robustness of accuracy and performance
  - reproducibility, fault tolerance, and auditability
GrADSoft Architecture

- Goal: **reliable performance under varying load**

GrADS Project (NSF NGS): Berman, Chien, Cooper, Dongarra, Foster, Gannon Johnsson, Kennedy, Kesselman, Mellor-Crummey, Reed, Torczon, Wolski
GrADSoft Architecture

**Program Preparation System**

- Source Application
- Whole-Program Compiler
- Configurable Object Program
- Performance Feedback
- Libraries

**Execution Environment**

- Real-time Performance Monitor
- Service Negotiator
- Scheduler
- Dynamic Optimizer
- Grid Runtime System

**Key Components**

- Whole-program Compiler
- Configurable Object Program
- Performance Feedback
- Real-time Performance Monitor
- Service Negotiator
- Scheduler
- Dynamic Optimizer
- Grid Runtime System
High Level Programming System

- Built on Libraries Coded by Professionals
  - Included mapping strategies and cost models
  - High level knowledge of integration strategies

- Integration Produces Configurable Object Program
  - Integrated mapping strategy and cost model
  - Performance enhanced through context-dependent variants
  - Context includes potential execution platform

- Dynamic Optimizer Performs Final Binding
  - Implements mapping strategy
  - Chooses machine-specific variants
  - Inserts monitoring probes

- Strategy: move compilation overhead to PSE-generation time
Configurable Object Program

- **Representation of the Application**
  - Supporting dynamic reconfiguration and optimization for distributed targets, may include
    - Program intermediate code
    - Annotations from the compiler
      - mapping strategy and performance model
    - Historical information (run profile to now)

- **Mapping strategies**
  - Aggregation of data regions (submeshes) or tasks
  - Definition of parameters for algorithm selection

- **Challenge: synthesis of performance models**
  - User input, especially associated with libraries
  - Synthesize different components, scale models to problem size
Execution Cycle

- Configurable Object Program is presented
  - Space of feasible resources must be defined
  - Mapping strategy and performance model provided
- Service Negotiator solicits acceptable resource collections
  - Performance model is used to evaluate each
  - Best match is selected and contracted for
- Execution begins
  - Dynamic optimizer tailors program to resources
    - Selects mapping strategy
    - Inserts sensors
- Contract monitoring is conducted during execution
  - Soft violation detection based on fuzzy logic
Performance Contracts

• At the Heart of the GrADS Model
  - Fundamental mechanism for managing mapping and execution

• What are they?
  - Mappings from resources to performance
  - Mechanisms for determining when to interrupt and reschedule

• Abstract Definition
  - Random Variable: $r(A,I,C,t_0)$ with a probability distribution
    - $A = \text{app}$, $I = \text{input}$, $C = \text{configuration}$, $t_0 = \text{time of initiation}$
    - Important statistics: lower and upper bounds (95% confidence)

• Challenge
  - When should a contract be (viewed as) violated?
    - Strict adherence balanced against cost of reconfiguration
Grids - Performance

- Performance tools for computational grids
  - Grid environments are dynamic
  - applications and computational resources are also dynamic
    - must adapt to sustain predictable levels of performance
    - prerequisite of adaptation is recognition of changing conditions

- Approach
  - performance contract
    - specifies application and resource commitments
  - application and execution signature models
    - predict application and resource behavior
  - monitoring and forecasting infrastructure
    - detects when actual and predicted behaviors do not match

- Contract specification model options
  - measurement and forecasts, compiler, library, and/or user
  - historical, current, and predicted data

Courtesy Dan Reed
GrADS: Grid Applications Development and Services

Given
- a set of resources (compute, network, I/O, ...)
- with certain capabilities (FLOP rate, latency, ...)
- for given application parameters (matrix size, image resolution, ...)

the application will
- exhibit a specified, measurable, and desirable performance
  - sustain F FLOPS/second, render R frames/second, ...

as predicted by the model(s) (global composition of models)

- Performance contracts specify a convolution of
  - application intrinsic behavior and system resource responses

- Monitoring infrastructure verifies contract fulfillment
  - performance sensors inserted/activated where needed
    - real-time measurement and forecasting
      - application, systems, resources (NWS, ...)
  - contract monitor detects when
    - actual and predicted behaviors do not match

Courtesy Dan Reed
Performance Signatures

- **Application intrinsic metrics**
  - description of application demands on resources
  - sample metrics
    - bytes/message or FLOPS/source statement
  - values are independent of execution platform
  - but they may depend on problem parameters

- **Execution space metrics**
  - reflect application demands and resource response to those demands
  - express rates of progress
  - sample metrics
    - instructions/second or messages/second
  - values are dependent on execution platform
  - quantify actual performance and may include application interplay

- **Application and execution signatures**
  - trajectories of values through N-dimensional metric space

Courtesy Dan Reed
Contract Monitoring

• Input:
  - Performance model
    • Integrated from a variety of sources: user, compiler, application experience, application signatures
  - Resources contracted for

• Trigger
  - Registration information from sensors installed in applications
    • Inserted by dynamic optimizer or user

• Output
  - Rule based contract monitor that decides when contract violation is serious enough to merit reconfiguration
    • Based on information from sensors
Grid Computing - Core Library

ScaLAPACK

- ScaLAPACK is a portable distributed memory numerical library
- Complete numerical library for dense matrix computations
- Designed for distributed parallel computing (MPP & Clusters) using MPI
- One of the first math software packages to do this
- Numerical software that will work on a heterogeneous platform
- Funding from DOE, NSF, and DARPA
- In use today by IBM, HP-Convex, Fujitsu, NEC, Sun, SGI, Cray, NAG, IMSL, ...
  - Tailor performance & provide support

Courtesy Jack Dongarra
Grid Computing - Core Library

ScaLAPACK

- ScaLAPACK is a portable distributed memory numerical library
- Complete numerical library for dense matrix computations
- Designed for distributed parallel computing (MPP & Clusters) using MPI
- One of the first math software packages to do this
- Numerical software that will work on a heterogeneous platform
- Funding from DOE, NSF, and DARPA
- In use today by IBM, HP-Convex, Fujitsu, NEC, Sun, SGI, Cray, NAG, IMSL, ...
- Tailor performance & provide support

Courtesy Jack Dongarra
Grid Computing – Library Use

- Download the package and auxiliary packages (like PBLAS, BLAS, BLACS, & MPI) to the machines.
- Write a SPMD program which
  - Sets up the logical 2-D process grid
  - Places the data on the logical process grid
  - Calls the numerical library routine in a SPMD fashion
  - Collects the solution after the library routine finishes
- The user must allocate the processors and decide the number of processes the application will run on
- The user must start the application
  - “mpirun -np N user_app”
    - Note: the number of processors is fixed by the user before the run, if problem size changes dynamically ...
- Upon completion, return the processors to the pool of resources

Courtesy Jack Dongarra
Grid Computing - Library Use

- Want to relieve the user of some of the tasks
- Make decisions on which machines to use based on the user's problem and the state of the system
  - Determinate machines that can be used
  - Optimize for the best time to solution
  - Distribute the data on the processors and collections of results
  - Start the SPMD library routine on all the platforms
  - Check to see if the computation is proceeding as planned
    - If not perhaps migrate application

Courtesy Jack Dongarra
User makes a sequential call to a numerical library routine. The Library Routine has “crafted code” which invoke other components.

Slide courtesy Jack Dongarra
GrADS Library Sequence

Library Routine calls a grid based routine to determine which resources are possible for use. The Resource Selector returns a “bag of processors” (coarse grid) that are available.

Slide courtesy Jack Dongarra
GrADS Library Sequence

The Library Routine calls the Performance Modeler to determine the best set of processors to use for the given problem. May be done by evaluating a formula or running a simulation. May assign a number of processes to a processor. At this point have a fine grid.
The Library Routine calls the Contract Development routine to commit the fine grid for this call. A performance guarantee is generated.
GrADS Library Sequence

User → Library Routine → Resource Selector

- App Launcher
- Contract Development
- Performance Model

"mpirun -machinefile fine_grid grid_linear_solve"

Slide courtesy Jack Dongarra

August 19, 2003
## Grids - Library Evaluation

### Experimental Hardware / Software Grid

<table>
<thead>
<tr>
<th>MacroGrid Testbed</th>
<th>TORC</th>
<th>CYPHER</th>
<th>OPUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Cluster 8 Dual Pentium III</td>
<td>Cluster 16 Dual Pentium III</td>
<td>Cluster 8 Pentium II</td>
</tr>
<tr>
<td>OS</td>
<td>Red Hat Linux 2.2.15 SMP</td>
<td>Debian Linux 2.2.17 SMP</td>
<td>Red Hat Linux 2.2.16</td>
</tr>
<tr>
<td>Memory</td>
<td>512 MB</td>
<td>512 MB</td>
<td>128 or 256 MB</td>
</tr>
<tr>
<td>CPU speed</td>
<td>550 MHz</td>
<td>500 MHz</td>
<td>265 – 448 MHz</td>
</tr>
<tr>
<td>Network</td>
<td>Fast Ethernet (100 Mbit/s) (3Com 3C905B) and switch (BayStack 350T) with 16 ports</td>
<td>Gigabit Ethernet (SK-9843) and switch (Foundry FastIron II) with 24 ports</td>
<td>Myrinet (LANai 4.3) with 16 ports each</td>
</tr>
</tbody>
</table>

- Globus version 1.1.3
- Autopilot version 2.3
- NWS version 2.0.pre2
- MPICH-G version 1.1.2
- ScaLAPACK version 1.6
- ATLAS/BLAS version 3.0.2
- BLACS version 1.1
- PAPI version 1.1.5
- GrADS’ “Crafted code”

Independent components being put together and interacting

---

Courtesy Jack Dongarra
Arrays of Values Generated by Resource Selector

- Clique based
  - 2 @ UT, UCSD, UIUC
  - Full at the cluster level and the connections (clique leaders)
  - Bandwidth and Latency information looks like this.
  - Linear arrays for CPU and Memory

![Table](image)
Grids - Performance Models

Speed = 60% of the peak

Latency in msec

Bandwidth in Mb/s

This is for a refined grid

Courtesy Jack Dongarra
Grids - Library Evaluation

Courtesy Jack Dongarra

<table>
<thead>
<tr>
<th>Grid</th>
<th>Non-Grid</th>
<th>Grid</th>
<th>Non-Grid</th>
<th>Grid</th>
<th>Non-Grid</th>
<th>Grid</th>
<th>Non-Grid</th>
<th>Grid</th>
<th>Non-Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time for Application Execution</td>
<td>Time for processes spawning</td>
<td>Time for NWS retrieval</td>
<td>Time for MDS retrieval</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ratio: 46.12</td>
<td>Ratio: 15.03</td>
<td>Ratio: 2.25</td>
<td>Ratio: 1.52</td>
<td>Ratio: 1.29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Grids - Library Evaluation

Courtesy Jack Dongarra
Matrix of size 30,000
- 7.2 GB for the data
- 32 processors to choose from UIUC and UT
  - Not all machines have 512 MBs, some little as 128 MBs
- PM chose 17 machines in 2 clusters from UT
- Computation took 84 minutes
  - 3.6 Gflop/s total
  - 210 Mflop/s per processor
  - ScALAPACK on a cluster of 17 processors would get about 50% of peak
  - Processors are 500 MHz or 500 Mflop/s peak
  - For this grid computation 20% less than ScALAPACK
Sample Application - Cactus

Numerical Relativity & Cactus

- Numerical simulation of extreme astrophysical events: colliding black holes, neutron stars, ...
  - Understand physics; predict gravitational wave forms
  - Relativistic effects => Einstein eqns
    - Computationally intensive (can be 1000s flops/grid point)
    - 3-D simulations only recently possible

- Cactus = modular, portable framework for parallel, multidimensional simulations
  - Construct codes by linking
    - Small core (flesh): mgmt services
    - Selected modules (thorns): Num. methods, grids & domain decomps, viz, steering, etc.
  - Custom linking/configuration tools

Courtesy Ed Seidel

LIGO gravitational wave observatory

August 19, 2003
NGSSC School on Grid Computing
Lennart Johnsson
Cactus on the Grid

- Application behaviors in a Grid environment:
  - Identify fastest/cheapest/best resources
  - Configure for efficient execution
  - Detect need for new resources or behaviors (e.g., new subtasks, resource slowdown, new appln regime, new resource available)
  - Adapt, and/or discover new resources; invoke subtasks on new resources and/or migrate

- Cactus thorns for management of appln behavior & resource use
  - Heterogeneous resources, e.g.:
    - Irregular decomp.; comms scheduling for comp/comm overlap
    - Variable halo for managing message size
    - Msg compression (comp/comm tradeoff)
  - Dynamic resource behaviors/demands, e.g.:
    - Perf monitoring, contract violation detection
    - Dynamic resource discovery, subtask spawning, migration
    - User notification and steering

Courtesy Ed Seidel
Cactus on the Grid

SDSC IBM SP
1024 procs
5x12x17 = 1020

NCSA Origin Array
256+128+128
5x12x(4+2+2) = 480

- Solved EE's for gravitational waves (real code)
  - Tightly coupled, communications required through derivatives
  - Must communicate 30MB/step between machines
  - Time step take 1.6 sec

- Used 10 ghost zones along direction of machines: communicate every 10 steps

- Compression/decomp. on all data passed in this direction

- Achieved 70-80% scaling, ~200GF (only 14% scaling without tricks)

Courtesy Ed Seidel
Cactus - Migration basis

- **Tequila thorn**
  - Contract monitor driven by three user-controllable parameters
    - Time quantum for “time per iteration”
    - % degradation in time per iteration (relative to prior average) before noting violation
    - Number of violations before migration
  - Communicates with resource monitor via ClassAd-based protocol
    - Specify resource requirements & performance model
    - Can request synchronous or asynchronous notification
  - Generates checkpoint and initiates migration

- **Resource selector**
  - Uses Globus Toolkit MDS-2 mechanisms to discover and monitor resources
  - Implements “cluster matching” algorithm to detect suitable clusters

Courtesy Ed Seidel, Ian Foster
Cactus - Job Migration

- Migrate to "faster/ cheaper" system
  - When better system discovered
  - When requirements change
  - When characteristics change (e.g., competition)
  - On user request
- Tests most elements of Cactus & GrADS
- Evaluate on GrADS testbed

- Architecture involves new Cactus thorn
  - Resource selector detects available resources and determines when to migrate
  - Application manager orchestrates migration
  - Globus Toolkit substrate for resource discovery, allocation, management

Courtesy Ed Seidel, Ian Foster
Cactus - Migration Architecture

(0) Possible user input
(1) Adapt. request
(1') Resource notification
(2) Resource request
(3) Write checkpoint
(4) Migration request
(5) Cactus startup
(6) Load code
(7) Read checkpoint

Application Manager
GrADS Resource Selector
Grid Information Service

Compute resource
Compute resource
Storage resource
Storage resource
Code repository
Code repository

Cactus "flesh"
"Tequila" Thorn
Appln & other thorns

Courtesy Ed Seidel, Ian Foster
Cactus - Migration example

Courtesy Ed Seidel, Ian Foster
Bioimaging

A. Virus

B. Neuron

C. Mouse Brain

D. Human Brain

<table>
<thead>
<tr>
<th>Raw Data</th>
<th>Reconstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Image]</td>
<td>[Image]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A. Virus</th>
<th>B. Neuron</th>
<th>C. Mouse Brain</th>
<th>D. Human Brain</th>
</tr>
</thead>
<tbody>
<tr>
<td>100nm</td>
<td>500μm</td>
<td>Object Size</td>
<td>10mm</td>
</tr>
<tr>
<td>5Å</td>
<td>500nm</td>
<td>Resolution</td>
<td>3μm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>250mm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1mm</td>
</tr>
</tbody>
</table>
Imaging Instruments

- Synchrotrons
  - Macromolecular Complexes, Organelles, Cells
- Microscopes
- Magnetic Resonance Imagers
  - Organs, Organ Systems, Organisms
- Molecular Structures
No. of Particles Needed for 3-D Reconstruction

<table>
<thead>
<tr>
<th>Resolution</th>
<th>8.5 Å</th>
<th>4.5 Å</th>
</tr>
</thead>
<tbody>
<tr>
<td>B = 100 Å²</td>
<td>6,000</td>
<td>5,000,000</td>
</tr>
<tr>
<td>B = 50 Å²</td>
<td>3,000</td>
<td>150,000</td>
</tr>
</tbody>
</table>

8.5 Å Structure of the HSV-1 Capsid
EM imaging

EMEN Database
- Archival
- Data Mining
- Management

Vitrification Robot

Project
- 200 – 10,000+ micrographs
- 10,000 – 10,000,00 particles
- 10k – 1,000k pixels/particle
- Up to hundreds of PFlops

EMAN

Particle Selection Power Spectrum Analysis

Initial 3D Model

Classify Particles

Reproject 3D Model

Align Average Deconvolute

Build New 3D Model

EMAN Image

Micrographs
- 4 - 64 Mpixels, 16-bit (8 – 128 MB)
- 100 – 200/day per lab
- 10 – 1,000 particles per micrograph
- Several TB/yr

EM imaging

NGSSC School on Grid Computing

Lennart Johnsson
NGSSC School on Grid Computing
Lennart Johnsson
Grid Application Software

• Managing Data, Codes and Resources Securely
• Performance
SimDB: A Grid Based Problem Solving Environment
Molecular Dynamics

MD SIMULATION:

20 ps Thermalization
500 ps Data Acquisition
Whole Flexible HIV-1 IN
CHARMM Force Field
PME Methods
(No Electrostatic Cutoff)

Jim Briggs
University of Houston
Molecular Dynamics Simulations

- raw trajectory
  - solute
  - solvent
  - 10^3 - 10^6 particles
  - 10^4 - 10^7 timesteps

- processed data
  - 10^1 - 10^7 values
  - analysis

- final processing
  - 1 - 1000 MB
  - 10^1 - 10^2 CPU seconds

- final output
  - 0.1 - 100 GB
  - 10^2 - 10^6 CPU seconds
  - 1 - 1000 KB

August 19, 2003
NGSSC School on Grid Computing
Lennart Johnsson
## Trajectory Properties

<table>
<thead>
<tr>
<th>Function</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport coefficients</td>
<td>Scalar</td>
</tr>
<tr>
<td>Normal modes</td>
<td>Set of scalar values</td>
</tr>
<tr>
<td>Solvent residence times</td>
<td>Set of scalar values</td>
</tr>
<tr>
<td>Average structures</td>
<td>3 x 1D coordinates</td>
</tr>
<tr>
<td>Animation</td>
<td>3 x 1D coordinates x N frames</td>
</tr>
<tr>
<td>Radial distribution functions</td>
<td>1D histogram</td>
</tr>
<tr>
<td>Energy profiles</td>
<td>1D or 2D histogram</td>
</tr>
<tr>
<td>Solvent densities</td>
<td>2D or 3D histogram</td>
</tr>
<tr>
<td>Angles/distances between solute groups</td>
<td>1D time series</td>
</tr>
<tr>
<td>Atom-atom distances</td>
<td>1D time series</td>
</tr>
<tr>
<td>Dihedral angles</td>
<td>1D time series</td>
</tr>
<tr>
<td>Root mean square deviations</td>
<td>1D time series</td>
</tr>
<tr>
<td>Simple thermodynamic properties</td>
<td>1D time series</td>
</tr>
<tr>
<td>Time correlation functions</td>
<td>1D time series</td>
</tr>
</tbody>
</table>
SimDB Workflow

- raw trajectory: 0.1 - 100 GB
- analysis: 10^6 - 10^7 timesteps, 10^3 - 10^6 CPU seconds
- processed data: 1 - 10^3 MB, 10^3 - 10^2 CPU seconds
- final output: 1 - 1000 KB

NGSSC School on Grid Computing

Lennart Johnsson
SimDB

University of Houston

Grid Services

Logon  User Profile  Submit Job  View Jobs  File Transfer  Resource Info  Queue Info

Job Submission

Intended as a "very" simple example of running a simple command. Good examples to try include executable "Hello" and arguments ".-d" or accessible "Hello.exe" etc.

- Global Queue keeper
- SSH scheme of SSH
- Wait for job output?
- E-mail me when job has completed?

Job Name: 
Hostname: seu2.cs.uh.edu
Jobmanager: fork
Executable: 
Arguments: 

Run Job
SimDB Data Access

User Client
- Query SimDB with trajectory data search criteria
- Trajectory data selection list
- Select Trajectory data from list
- Propagation selection list mean for previously selected Trajectory data
- Select property from list
- Processed data selection list mean
- Select processed data from list
- Subdivision selection list mean for processed data
- Select subdivision from list
- Present domain specification
- Specify new range limits
- List of filters that can be applied on the final data selection
- Select filter to be applied
- Login prompt
- User Name and Password
- Report User Resource options
- Choose resource

Application Server
- Query with trajectory data search criteria
- Available trajectory data list
- Query for property list using the selected trajectory data ID
- Available property list
- Query for processed data list using the selected trajectory data ID and property ID
- Available processed data list
- Query for subdivision list using the selected processed data ID
- Available subdivision list
- Query for domain description using subdivision
- Dimension / units / limits of Domain

Central Metadata Server
- Grid Computing Environment
- Grid Computing Environment
- Notify User Of Job Completion
- Logoff
- Query CMS for User profile
- Return user profile
- Get current status of User’s resource
- Get status
- Update User profile
- User profile Updated

GrADS/SIR/CM
- Resource reservation
- Schedule resources
- Available trajectory data list
- Query with trajectory data search criteria
- Available property list
- Available processed data list
- Get final data selection
- Send contract info
- Execute Job on User resource
- Job completed
- Send contract info
- Transfer data
- Data transferred
- User profile Updated

Prepared Data Server
- Data transferred
- Transfer data
- Send contract info
- Execute Job
- Job completed
- Send contract info
- Transfer data
- Data transferred
- Send contract info

Application Repository Server
- Code Transferred
- Send contract info
- Execute Job
- Job completed
- Send contract info
- Transfer data
- Data transferred
- Send contract info

User Resource 1
- Transfer data
- Data transferred
- Send contract info
- Execute Job
- Job completed
- Send contract info
- Transfer data
- Data transferred
- Send contract info

User Resource 2
- Transfer data
- Data transferred
- Send contract info
- Execute Job
- Job completed
- Send contract info
- Transfer data
- Data transferred
- Send contract info
SimDB Administration
GrADSoft Architecture

Program Preparation System

Execution Environment

Source Application
Whole-Program Compiler
Configurable Object Program
Performance Feedback

Software Components
Performance Problem
Service Negotiator
Scheduler
Real-time Performance Monitor
Dynamic Optimizer

Libraries

Grid Runtime System

Negotiation

Real-time Performance Monitor

August 19, 2003
NGSSC School on Grid Computing
Lennart Johnsson
Adaptive Software
Challenges

• Algorithmic
  – Multiple data structures and their interaction
  – Unfavorable data access pattern (big $2^n$ strides)
  – High efficiency of the algorithm
    • low floating-point v.s. load/store ratio
  – Additions/multiplications unbalance
• Version explosion
  – Verification
  – Maintenance
Challenges

• Diversity of execution environments
  – Growing complexity of modern microprocessors.
    • Deep memory hierarchies
    • Out-of-order execution
    • Instruction level parallelism
  – Growing diversity of platform characteristics
    • SMPs
    • Clusters (employing a range of interconnect technologies)
    • Grids (heterogeneity, wide range of characteristics)

• Wide range of application needs
  – Dimensionality and sizes
  – Data structures and data types
  – Languages and programming paradigms
Opportunities

- Multiple algorithms with comparable numerical properties for many functions
- Improved software techniques and hardware performance
- Integrated performance monitors, models and databases
- Run-time code construction
Approach

• Automatic algorithm selection – polyalgorithmic functions
  – Entirely different algorithms, exploit decomposition of operations,…
• Code generation from high-level descriptions
• Extensive application independent compile-time analysis
• Integrated performance modeling and analysis
• Run-time application and execution environment dependent composition
• Automated installation process
Performance Tuning Methodology

- **Input Parameters**
  - System specifics
  - User options

- **UHFFT Code generator**

- **Library of FFT modules**

- **Performance database**

**Installation**

**Run-time**

- **Input Parameters**
  - Size, dim., ...

- **Initialization**
  - Select best plan

- **Execution**
  - Calculate one or more FFTs
The UHFFT

- Program preparation at installation (platform dependent)
- Integrated performance models (in progress) and data bases
- Algorithm selection at run-time from set defined at installation
- Automatic multiple precision constant generation
- Program construction at run-time based on application and performance predictions
The Fast Fourier Transform

- Application of $W_n$ requires $O(n^2)$ operations.
- Fast algorithms $\iff$ sparse factorizations of $W_n$

$$W_n = A_1 A_2 \ldots A_k$$

where $A_i$ are sparse ($O(n)$ operations) and $k = \log(n)$.
- Application of $W_n$ can be evaluated in $O(n \log(n))$ ops.
- Sparse factorization is not unique.
- Many different ways to factor the same $W_n$. 
Sparse Factorization Algorithms

- Rader’s Algorithm
- Prime Factor Algorithm
- Split-radix
- Mixed-radix (Cooley-Tukey)
Notation

• Simple way to describe highly structured matrices
• Tensor (Kronecker) product

\[
A \otimes B = \begin{pmatrix}
    a_{00}B & a_{01}B & \ldots & a_{0n-1}B \\
    a_{10}B & a_{11}B & \ldots & a_{1n-1}B \\
    \vdots & \vdots & \ddots & \vdots \\
    a_{n-10}B & a_{n-11}B & \ldots & a_{n-1n-1}B
\end{pmatrix}
\]

\[A = \text{scaling matrix}, \quad B = \text{blocking matrix}\]

• Direct sum notation:

\[
A \oplus B = \begin{pmatrix}
    A & 0 \\
    0 & B
\end{pmatrix}
\]
Prime Factor Algorithm

When $n = rq$, and $\gcd(r, q) = 1$ it is possible to reduce the number of operations by using the PFA (no twiddle factor multiplication).

$$W_n = \Pi_1 (W_r \otimes I_q)(I_r \otimes W_q)\Pi_2$$

$$= \Pi_1 (W_r \otimes W_q)\Pi_2 ,$$

where $\Pi_1, \Pi_2$ are permutation matrices.
Split-radix Algorithm

- When $n = 2^k$ the most efficient algorithm.
- Two levels of radix-2 splitting.

$$W_n = (W_2 \otimes I_{n/2})D_{2,n/2} (I_2 \otimes W_{n/2})\Pi_{n,2}$$

- When $n = 2q = 4p$ we can write

$$W_n = B_{SR} (W_q \oplus W_p \oplus W_p)\Pi_{n,q,2},$$

$$B_{SR} = B_a B_m,$$

$$B_a = (W_2 \otimes I_q)[I_q \oplus (\tilde{W}_2 \otimes I_p)],$$

$$B_m = I_q \oplus \Omega_{n,p} \oplus \Omega_{n,p}^3,$$

$$\tilde{W}_2 = (1 \oplus -i)W_2,$$

$$\Pi_{n,q,2} = (I_q \oplus \Pi_{q,2})\Pi_{n,2}$$
Mixed-radix Splitting

When \( n = rq \), \( W_n \) can be written as

\[
W_n = (W_r \otimes I_q)D_{r,q}(I_r \otimes W_q)\Pi_{n,r},
\]

where \( D_{r,q} \) is a diagonal twiddle factor matrix

\[
D_{r,q} = I_q \oplus \Omega_{n,q} \oplus \cdots \oplus \Omega_{n,q}^{r-1},
\]

\[
\Omega_{n,q} = 1 \oplus \omega_n \oplus \cdots \oplus \omega_n^{r-1},
\]

and \( \Pi_{n,r} \) is a mod-\( r \) sort permutation matrix.
Algorithm Selection Logic

if \( n \leq 2 \) DFT
else if \( n \) is prime use Rader’s algorithm
else {
    Chose factor \( r \) of \( n \)
    if \( r \) and \( n/r \) are coprime use PFA
    else if \( n \) is divisible by \((r^2)\) and \( n > r^3 \)
        use Split-Radix algorithm
    else use Mixed-radix algorithm
}
Example $N = 6$

- $\text{FFTPrimeFactor } n = 6, \ r = 3, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{FFTRader } n = 3, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Inverse}, \ \text{rot} = 1$
- $\text{FFTRader } n = 3, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Inverse}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
- $\text{DFT } n = 2, \ r = 2, \ \text{dir} = \text{Forward}, \ \text{rot} = 1$
UHFFT Architecture

- Library of FFT Modules
- Initialization Routines
- Execution Routines
- Utilities

- FFT Code Generator
- Mixed-Radix (Cooly-Tukey)
- Prime Factor Algorithm
- Split-Radix Algorithm
- Rader's Algorithm

Key:
- Fixed library code
- Generated code
- Code generator

Funded in part by the Alliance (NSF) and LACSI (DoE)
Optimization Strategy

- High level (performed at application initialization)
  - Selection of the optimal factorization using preoptimized codelets
  - Generation of an execution plan

- Low level
  - Selected set of codelets
  - Selected codelets automatically generated and highly optimized by a special-purpose compiler
The UHFFT: Code Generation

• Structure
  • Algorithm abstraction
  • Optimization
  • Generation of a DAG
  • Scheduling of instructions
  • Unparsing

• Implementation
  • Code generator is written in C
    • Speed, portability and installation tuning
    • Highly optimized straight line C code
    • Generates FFT codelets of arbitrary size, direction, and rotation
The UHFFT: Code Generation (cont’d)

- Basic structure is an *Expression*
  - Constant, variable, sum, product, sign change, …

- Basic functions
  - Expression sum, product, assign, sign change, …

- Derived structures
  - Expression vectors, matrices and lists

- Higher level functions
  - Matrix vector operations
  - FFT specific operations

- Algorithms currently supported
  - Rader (two versions), PFA, Split-radix, Mixed-radix
The UHFFT: Code Generation Mixed-Radix Algorithm

Equation: \( W_n = (W_r \otimes I_m)D_{r,m} (I_r \otimes W_m)\Pi_{n,k} \)

Is implemented as:

```c
/*
* FFTMixedRadix() Mixed-radix splitting.
* Input:
*    r    radix,
*    dir, rot direction and rotation of the transform,
*    u    input expression vector.
*/
ExprVec *FFTMixedRadix(int r, int dir, int rot, ExprVec *u)
{
    int m, n = u->n, *p;
    m = n/r;
    p = ModRSortPermutation(n, r);
    u = FFTxI(r, m, dir, rot,
             TwiddleMult(r, m, dir, rot,
                        IxFFT(r, m, dir, rot, PermuteExprVec(u, p))));
    free(p);
    return u;
}
```
The UHFFT: Performance Modeling

- Analytic models
  - Cache influence on library codes
  - Performance measuring tools (PCL, PAPI)
  - Prediction of composed code performance
  - Updated from execution experience
- Data base
  - Library codes. Recorded at installation time
  - Composed codes. Recorded and updated for each execution.
The UHFFT: Execution Plan Generation

- Optimal plan search options
  - Exhaustive
  - Recursive
  - Empirical

- Algorithms used
  - Rader (FFTW, UHFFT)
  - PFA (UHFFT)
  - Split-radix (UHFFT)
  - Mixed-radix (FFTW, SPIRAL, UHFFT)
### Characteristics of Some Processors

<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock frequency</th>
<th>Peak Performance</th>
<th>Cache structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Pentium IV</td>
<td>1.8 GHz</td>
<td>1.8 GFlops</td>
<td>L1: 8K+8K, L2: 256K</td>
</tr>
<tr>
<td>AMD Athlon</td>
<td>1.4 GHz</td>
<td>1.4 GFlops</td>
<td>L1: 64K+64K, L2: 256K</td>
</tr>
<tr>
<td>Intel Itanium</td>
<td>800 MHz</td>
<td>3.2 GFlops</td>
<td>L1: 16K+16K, L2: 92K, L3: 2-4M</td>
</tr>
<tr>
<td>IBM Power3/4</td>
<td>375 MHz</td>
<td>1.5 GFlops</td>
<td>L1: 64K+32K, L2: 1-16M</td>
</tr>
<tr>
<td>HP PA 8x00</td>
<td>750 MHz</td>
<td>3 GFlops</td>
<td>L1: 1.5M + 0.75M</td>
</tr>
<tr>
<td>Alpha EV67/68</td>
<td>833 MHz</td>
<td>1.66 GFlops</td>
<td>L1: 64K+64K, L2: 4M</td>
</tr>
<tr>
<td>MIPS R1x000</td>
<td>500 MHz</td>
<td>1 GFlop</td>
<td>L1: 32K+32K, L2: 4M</td>
</tr>
</tbody>
</table>
Codelet efficiency

Intel PIV 1.8 GHz

AMD Athlon 1.4 GHz

PowerPC G4 867 MHz
Power3 plan performance

MFLOPS

Plan

222 MHz
888 Mflops
IBM Power3 222 MHz
Execution Plan Comparison

\[ n = 2520 \text{ (PFA Plan)} \]
**HP zx1 Chipset**

**Features:**
- 2-way and 4-way
- Low latency connection to the DDR memory (112 ns)
  - Directly (112 ns latency)
  - Through (up to 12) scalable memory expanders (+25 ns latency)
- Up to 64 GB of DDR today (256 in the future)
- AGP 4x today (8x in the future versions)
- 1-8 I/O adapters supporting
  - PCI, PCI-X, AGP
<table>
<thead>
<tr>
<th>Processor</th>
<th>Clock frequency</th>
<th>Peak Performance</th>
<th>Cache structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Itanium</td>
<td>800 Mhz</td>
<td>3.2 GFlops</td>
<td>L1: 16K+16K (Data+Instruction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: 92K, L3: 2-4M (off-die)</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>900 Mhz</td>
<td>3.6 GFlops</td>
<td>L1: 16K+16K (Data+Instruction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: 256K, L3: 1.5M (on-die)</td>
</tr>
<tr>
<td>Intel Itanium 2</td>
<td>1000 Mhz</td>
<td>4 GFlops</td>
<td>L1: 16K+16K (Data+Instruction)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: 256K, L3: 3M (on-die)</td>
</tr>
<tr>
<td>Sun UltraSparc-III</td>
<td>750 Mhz</td>
<td>1.5 GFlops</td>
<td>L1: 64K+32K+2K+2K (Data+Instruction+Pre-fetch+Write)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: up to 8M (off-die)</td>
</tr>
<tr>
<td>Sun UltraSparc-III</td>
<td>1050 Mhz</td>
<td>2.1 GFlops</td>
<td>L1: 64K+32K+2K+2K (Data+Instruction+Pre-fetch+Write)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>L2: up to 8M (off-die)</td>
</tr>
</tbody>
</table>
# Memory Hierarchy

<table>
<thead>
<tr>
<th></th>
<th>Itanium-2 (McKinley)</th>
<th>Itanium</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size:</strong></td>
<td>16KB + 16KB</td>
<td>16KB + 16KB</td>
</tr>
<tr>
<td><strong>Line size/Associativity:</strong></td>
<td>64B/4-way</td>
<td>32B/4-way</td>
</tr>
<tr>
<td><strong>Latency:</strong></td>
<td>1 cycle</td>
<td>1 cycle</td>
</tr>
<tr>
<td><strong>Write Policies:</strong></td>
<td>Write through, No write allocate</td>
<td>Write through, No write allocate</td>
</tr>
<tr>
<td><strong>Integer Latency:</strong></td>
<td>Min 5 cycles</td>
<td>Min 6 cycles</td>
</tr>
<tr>
<td><strong>FP Latency:</strong></td>
<td>Min 6 cycles</td>
<td>Min 9 cycles</td>
</tr>
<tr>
<td><strong>Write Policies:</strong></td>
<td>Write back, write allocate</td>
<td>Write back, write allocate</td>
</tr>
<tr>
<td><strong>Bandwith:</strong></td>
<td>32B/cycle</td>
<td>16B/cycle</td>
</tr>
<tr>
<td><strong>Line size/Associativity:</strong></td>
<td>128B/12-way</td>
<td>64B/4-way</td>
</tr>
<tr>
<td><strong>Integer Latency:</strong></td>
<td>Min 12 cycles</td>
<td>Min 21 cycles</td>
</tr>
<tr>
<td><strong>FP Latency:</strong></td>
<td>Min 13 cycles</td>
<td>Min 24 cycles</td>
</tr>
<tr>
<td><strong>Bandwith:</strong></td>
<td>32B/cycle</td>
<td>16B/cycle</td>
</tr>
</tbody>
</table>

**L1I and L1D**

**Unified L2**

**Unified L3**
UHFFT Maximum Forward Codelet Performance

- Intel Itanium 800 MHz
- Intel Itanium 2 900 MHz
- Sun UltraSparc-III 750 MHz

"MFlop/s"

Codelet size

August 19, 2003  NGSSC School on Grid Computing  Lennart Johnsson
UHFFT Average Forward Codelet Performance

- Intel Itanium 800 MHz
- Intel Itanium 2 900 MHz
- Sun UltraSparc-III 750 MHz

"MFlop/s" vs. Codelet size
Codelet Performance Radix-2

UHFFT Radix-2 Itanium 800 MHz, \( P_{avg} = 222.5 \)

UHFFT Radix-2 UltraSparc III 750 MHz, \( P_{avg} = 175.1 \)

UHFFT Radix-2 Itanium 2 (McKinley) 900 MHz, \( P_{avg} = 332.5 \)
Codelet Performance Radix-3

UHFFT Radix-3 Itanium 800 MHz, \( P_{avg} = 382.0 \)

UHFFT Radix-3 UltraSparc III 750 MHz, \( P_{avg} = 296.2 \)

UHFFT Radix-3 Itanium 2 (McKinley) 900 MHz, \( P_{avg} = 491.5 \)
Codelet Performance Radix-4
Codelet Performance Radix-5

UHFFT Radix-5 Itanium 800 MHz, Pavg = 613.3

UHFFT Radix-5 UltraSparc III 750 MHz, Pavg = 299.4

UHFFT Radix-5 Itanium 2 (McKinley) 900 MHz, Pavg = 926.0

Output stride  Input stride

Output stride  Input stride

Output stride  Input stride
Codelet Performance Radix-6

UHFFT Radix-6 Itanium 800 MHz, Pavg = 905.3

UHFFT Radix-6 UltraSparc III 750 MHz, Pavg = 390.7

UHFFT Radix-6 Itanium 2 (McKinley) 900 MHz, Pavg = 1215.5
Codelet Performance Radix-7

UHFFT Radix-7 Itanium 900 MHz, Pavg = 596.9

UHFFT Radix-7 UltraSparc III 750 MHz, Pavg = 17

UHFFT Radix-7 Itanium 2 (McKinley) 900 MHz, Pavg = 1179.9
Codelet Performance Radix-8

UHFFT Radix-8 Itanium 800 MHz, Pavg = 717.7

UHFFT Radix-8 UltraSparc III 750 MHz, Pavg = 349.7

UHFFT Radix-8 Itanium 2 (McKinley) 900 MHz, Pavg = 1590.1
Codelet Performance Radix-9

UHFFT Radix-9 Itanium 800 MHz, $P_{avg} = 671.5$

UHFFT Radix-9 UltraSparc III 750 MHz, $P_{avg} = 271.7$

UHFFT Radix-9 Itanium 2 (McKinley) 900 MHz, $P_{avg} = 1343.5$
Codelet Performance Radix-16

UHFFT Radix-16 Itanium 900 MHz, Pavg = 905.3

UHFFT Radix-16 UltraSparc TII 750 MHz, Pavg = 297.8

UHFFT Radix-16 Itanium 2 (McKinley) 900 MHz, Pavg = 1921.1
Codelet Performance Radix-32

- UHFFT Radix-32 Itanium 900 MHz, $P_{avg} = 943.1$
- UHFFT Radix-32 UltraSparc T1 750 MHz, $P_{avg} = 265.9$
- UHFFT Radix-32 Itanium 2 (McKinley) 900 MHz, $P_{avg} = 1676.8$
Codelet Performance Radix-45

UHFFT Radix-45 Itanium 800 MHz, Pavg = 718.7

UHFFT Radix-45 UltraSparc III 750 MHz, Pavg = 266.1

UHFFT Radix-45 Itanium 2 (McKinley) 900 MHz, Pavg = 1305.4
Codelet Performance Radix-64

UHFFT Radix-64 Itanium 800 MHz, Pavg = 340.1

UHFFT Radix-64 UltraSparc III 750 MHz, Pavg = 229.0

UHFFT Radix-64 Itanium 2 (McKinley) 900 MHz, Pavg = 1122.8

Output stride
Input stride
Output stride
Input stride
Output stride
Input stride
GrADS Testbed

- Common services across testbeds and target execution environment
  - Smooth transition between macro and micro testbeds and production Grids

<table>
<thead>
<tr>
<th>Experimental GrADS Software Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core GrADS Software Environment</td>
</tr>
<tr>
<td>Core Middleware Services</td>
</tr>
<tr>
<td>MicroGrid Testbeds</td>
</tr>
<tr>
<td>MacroGrid Testbeds</td>
</tr>
<tr>
<td>Production Grids</td>
</tr>
</tbody>
</table>
GrADS Testbed

- Build on standard Globus deployment
- Specialized instrumentation and monitoring to drive GrADS software
  - Network Weather Service (measurement and prediction)
- Customized information services to capture execution space
  - GrADS testbed as a “virtual organization”
- Web based tools to disseminate information
  - Oriented towards users and administrators
- Additional services to support software distribution and other group activities
GrADS Testbed Web Interface

August 19, 2003  NGSSC School on Grid Computing  Lennart Johnsson
GrADS Testbed Web Interface
GrADS Testbed Services

- Need structure to coordinate distributed experiments
  - Distributed resource base, software base
  - Distributed experimenter base
- Provide project and experiment specific views of Grid
  - Collates and presents information about experiment resources in a uniform view
- More than just Grid stuff
  - Browsers, POC information, web pages, mailing lists etc
GrADS team

• Ken Kennedy
• Ruth Aydt/Celso Mendez
• Francine Berman/Henry Cassanov a
• Andrew Chien
• Keith Cooper
• Jack Dongarra
• Ian Foster
• Dennis Gannon
• Lennart Johnsson
• Carl Kesselman
• Dan Reed
• Richard Wolski
• Linda Torczon
• Many students

NSF funded under NGS