A Look at Some Ideas and Experiments

Jack Dongarra
University of Tennessee
and
Oak Ridge National Laboratory

Orientation

- The design of smart numerical libraries; libraries that can use the “best” available resources, analyze the data, and search the space of solution strategies to make optimal choices
- The development of “agent-based” methods for solving large numerical problems on both local and distant grids
- Development of a prototype framework based on standard components for building and executing composite applications
The Grid: Abstraction

- Semantically: the grid is nothing but abstraction
  - Resource abstraction
    - Physical resources can be assigned to virtual resource needs (matched by properties)
    - Grid provides a mapping between virtual and physical resources
  - User abstraction
    - Grid provides a temporal mapping between virtual and physical users

With The Grid...

- What performance are we evaluating?
  - Algorithms
  - Software
  - Systems

- What are we interested in?
  - Fastest time to solution?
  - Best resource utilization?
  - Lowest “cost” to solution?
  - Reliability of solution?
  - ...

NSF/NGS
GrADS - GrADSoft Architecture

- Goal: reliable performance on dynamically changing resources

PIs: Ken Kennedy, Fran Berman, Andrew Chein, Keith Cooper, JD, Ian Foster, Lennart Johnsson, Dan Reed, Carl Kesselman, John Mellor-Crummey, Linda Torczon & Rich Wolski
NSF/NGS
GrADS - GrADSoft Architecture

Goal: reliable performance on dynamically changing resources

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
To Use ScaLAPACK a User Must:

- 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

ScaLAPACK Grid Enabled

- Implement a version of a ScaLAPACK library routine that runs on the Grid.
  - Make use of resources at the user’s disposal
  - Provide the best time to solution
  - Proceed without the user’s involvement
- Make as few changes as possible to the numerical software.
- Assumption is that the user is already “Grid enabled” and runs a program that contacts the execution environment to determine where the execution should take place.
- Best time to solution
GrADS Numerical Library

- 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

Big Picture...

User has problem to solve (e.g. Ax = b)

Natural Data (A,b)  →  Natural Answer (x)

Middleware

Structured Data (A',b')  →  Structured Answer (x')

Application Library (e.g. LAPACK, ScaLAPACK, PETSc,...)
Numerical Libraries for Grids

User

Stage data to disk

A  b

Library
Middle-ware
Numerical Libraries for Grids

Uses Grid infrastructure, i.e., Globus/NWS.
GrADS Library Sequence

- Has “crafted code” to make things work correctly and together.

Assumptions:
Autopilot Manager has been started and
Globus is there.

Resource Selector

- Uses MDS and NWS to build an array of values
  - 2 matrices (bw,lat) 2 arrays (cpu, memory available)
  - Matrix information is clique based
- On return from RS, Crafted Code filters information to use only machines that have the necessary software and are really eligible to be used.
Resource Selector Input

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

- Matrix of values are filled out to generate a complete, dense, matrix of values.
- At this point have a workable coarse grid.

- Know what is available, the connections, and the power of the machines

Uses NWS to collect information

ScaLAPACK Performance Model

\[ T(n, p) = C_f t_f + C_v t_v + C_m t_m \]

- \( C_f = \frac{2n^3}{3p} \) Total number of floating-point operations per processor
- \( C_v = (3 + \frac{1}{4} \log_2 p) \frac{n}{\sqrt{p}} \) Total number of data items communicated per processor
- \( C_m = n(6 + \log_2 p) \) Total number of messages
- \( t_f \) Time per floating point operation
- \( t_v \) Time per data item communicated
- \( t_m \) Time per message
Resource Selector/Performance Modeler

- Refines the course grid by determining the process set that will provide the best time to solution.
- This is based on dynamic information from the grid and the routines performance model.
- The PM does a simulation of the actual application using the information from the RS.
  - It literally runs the program without doing the computation or data movement.
- There is no backtracking in the Optimizer.
  - This is an area for enhancement and experimentation.
- Simulated annealing used as well

Performance Model Validation

<table>
<thead>
<tr>
<th>mem(MB)</th>
<th>Opus14</th>
<th>Opus13</th>
<th>Opus16</th>
<th>Opus15</th>
<th>Torc4</th>
<th>Torc6</th>
<th>Torc7</th>
</tr>
</thead>
<tbody>
<tr>
<td>speed</td>
<td>270</td>
<td>270</td>
<td>270</td>
<td>270</td>
<td>330</td>
<td>330</td>
<td>330</td>
</tr>
<tr>
<td>load</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>1.00</td>
<td>0.94</td>
<td>0.95</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Speed = performance of DGEMM (ATLAS)

<table>
<thead>
<tr>
<th>Latency in msec</th>
<th>Opus14</th>
<th>Opus13</th>
<th>Opus16</th>
<th>Opus15</th>
<th>Torc4</th>
<th>Torc6</th>
<th>Torc7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opus14</td>
<td>-1</td>
<td>0.24</td>
<td>0.29</td>
<td>0.20</td>
<td>83.78</td>
<td>83.78</td>
<td>83.78</td>
</tr>
<tr>
<td>Opus13</td>
<td>0.24</td>
<td>-1</td>
<td>0.24</td>
<td>0.23</td>
<td>83.78</td>
<td>83.78</td>
<td>83.78</td>
</tr>
<tr>
<td>Opus15</td>
<td>0.23</td>
<td>0.24</td>
<td>-1</td>
<td>0.20</td>
<td>83.78</td>
<td>83.78</td>
<td>83.78</td>
</tr>
<tr>
<td>Torc4</td>
<td>83.78</td>
<td>83.78</td>
<td>83.78</td>
<td>-1</td>
<td>0.31</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>Torc6</td>
<td>83.78</td>
<td>83.78</td>
<td>83.78</td>
<td>0.31</td>
<td>-1</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>Torc7</td>
<td>83.78</td>
<td>83.78</td>
<td>83.78</td>
<td>0.31</td>
<td>0.31</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Latency in msec

<table>
<thead>
<tr>
<th>Bandwidth in Mb/s</th>
<th>Opus14</th>
<th>Opus13</th>
<th>Opus16</th>
<th>Opus15</th>
<th>Torc4</th>
<th>Torc6</th>
<th>Torc7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opus14</td>
<td>248.83</td>
<td>247.51</td>
<td>247.30</td>
<td>246.34</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
</tr>
<tr>
<td>Opus13</td>
<td>244.54</td>
<td>-1</td>
<td>247.54</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
</tr>
<tr>
<td>Opus15</td>
<td>247.54</td>
<td>247.54</td>
<td>-1</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
</tr>
<tr>
<td>Torc4</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>81.96</td>
<td>-1</td>
<td>50.9</td>
</tr>
<tr>
<td>Torc6</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>50.9</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>Torc7</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>2.83</td>
<td>50.9</td>
<td>50.9</td>
<td>-1</td>
</tr>
</tbody>
</table>

This is for a refined grid
### 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 (3Com 3C905B) and switch (BayStack 350T) with 16 ports</td>
<td>Gigabit Ethernet (SK-9843) and switch (Foundry Fastfront 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

### Grid ScaLAPACK vs Non-Grid ScaLAPACK, Dedicated Torc machines

- Time for Application Execution
- Time for processes spawning
- Time for NWS retrieval
- Time for MDS retrieval

<table>
<thead>
<tr>
<th>N</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>
<th>Grid</th>
<th>Non-Grid</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10,000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N=600, NB=40, 2 torc proc. Ratio: 66.12
N=1500, NB=40, 4 torc proc. Ratio: 15.05
N=5000, NB=40, 6 torc proc. Ratio: 2.25
N=8000, NB=40, 8 torc proc. Ratio: 1.52
N=10,000, NB=40, 8 torc proc. Ratio: 1.29
ScaLAPACK - PDGESV - Using collapsed NWS query from UCSB
42 machine available, using mainly torc, cypher, msc clusters at UTK

Matrix Size - Nproc

ScaLAPACK across 3 Clusters
Largest Problem Solved

- Matrix of size 35,000
  - 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 27 machines in 3 clusters from UT
  - Computation took 87 minutes
    - 5.5 Gflop/s total
    - 205 Mflop/s per processor
    - Rule of thumb for ScaLAPACK is about 50% of theoretical peak
    - Processors are 500 MHz or 500 Mflop/s peak
    - For this grid computation 6% less than ScaLAPACK

LAPACK For Clusters

- Developing middleware which couples cluster system information with the specifics of a user problem to launch cluster based applications on the “best” set of resource available.

- Using ScaLAPACK as the prototype software
NetSolve Timing
NetSolve Timing

NetSolve System

NetSolve Timing

NetSolve System
Grid Data Movement

- Experiments
  - Comparing BW and times for moving data over a grid/WAN/LAN using 4 protocols
  - NetSolve, scp, Globus gridftp using 1 stream, gridftp using 16 streams

- Data
  - Floats [0-1) or ints (for NetSolve) in binary format, can be compressed about 10%

- Grid Sites
  - UTK msc (dual 933 MHz PIII, Linux)
  - UTK torc (dual 550MHz PIII, Linux)
  - UCSD mystere (1733 MHz Athlon XP, Linux)
  - UH mckinley (900 MHz Itanium IA64, Linux)

Details on the Protocols

- NetSolve
  - Uses untuned TCP sockets to move data

- Scp
  - Encrypts data, and compresses by default (gzip –6)

- Globus-url-copy
  - Provides data authentication, GSI security, tcpip buffer and socket tuning, no compression/encryption

- Globus-16
  - Uses 16 parallel streams of globus-url-copy
Transfer maximums

- Measured using iperf w defaults
- UTK (msc) – UTK (torc)
  - BW to 89.4 Mb/sec = 11 MB/sec
  - BW back 93.6 Mb/sec
- UTK (msc) – UCSD (mystere)
  - BW to 7.96 Mb/sec = 1 MB/sec
  - BW back 8.00 Mb/sec
- UTK – UH (asymmetric)
  - BW to 9.4 Mb/sec = 1.2 MB/sec
  - BW back 16.4 Mb/sec = 2.0 MB/sec
- Globus-16 stream may exceed iperf 1-stream
  - UTK-UH BW back using 16 parallel threads 59 Mb/sec = 7.3 MB/sec

Details on the Protocols

- NetSolve
  - Only includes time to move data, no other overhead is included
- NetSolve-total
  - Includes all the parts of the NetSolve process, including overhead of contacting agent, etc
- Scp
  - Encrypts data, and compresses by default (gzip –6)
- Globus-url-copy
  - Provides data authentication, GSI security, tcpip buffer and socket tuning, no compression/encryption
- Globus-16
  - Uses 16 parallel streams of globus-url-copy
Bandwidth UTK – UTK

Over 5 runs and 2 days, little variance

Bandwidth UTK – UH

Over 5 runs and 2 days, little variance
Discussion

- NetSolve is good for all tested message sizes
  - This is unexpected, since NetSolve does no TCP tuning/etc to improve
  - Linux 2.4 has auto-tuning built into the kernel, which may explain the excellent results
  - We may get different results on Solaris, etc, but we could not compare with globus data transfer protocols, because GrADS does not have any other machine architectures currently
  - Multi-stream globus-16 can do better over WAN
Details on Timing

- Timing for NetSolve
  - Source code was instrumented and precise timings for each operation were obtained

- Timing for globus and scp
  - Perl script was used with a HiRes timer
    ```perl
    $start = time()
    open(CMD, "globus-url-copy file:/file
    gsiftp://remotesite/file" |)
    while (CMD) {}
    close(CMD)
    $end = time()
    return $end -$start
    
    - Same for scp command
Image Processing

- Image Segmentation
- Goal is to classify pixels into predefined classes which represent different features of the image. (e.g. Trees, Shadows, Roads, etc.)
- CPU intensive application
- Computations for each pixel class are independent of other classes.
- Prime candidate for distributed parallel implementation.
Parallel Image Segmentation

- MPI
- One node per pixel class
- Drastically reduces computation time since the pixel class computations are independent of each other.
- Final image is produced at master node by combining results from each worker node.

Image Segmentation as Remote Job

- **NetSolve Service**
  - MPI code is wrapped as a NetSolve service.
  - Clients pass input image and pixel class statistics to service.
  - Service launches MPI code to produce the output image.
  - Output image returned to the client as result.

- **Globus Job**
  - Client transfers input image to remote site (e.g. `globus-url-copy` file://inputfile.png `https://server/inputfile.png`).
  - Job and input parameters submitted to remote site using `globus-job-run`.
  - Resulting image is retrieved by user with `globus-url-copy`. 
Remote Processing Benchmarks

- Input data staging
- Job submission / Service request overhead
- Job compute time
- Result data retrieval

- The image processing code is run remotely using several different sizes for the input image.
- Computation parameters kept constant.
- Each image size is run 200 times on both NetSolve and Globus

- All remote jobs are run within the Torc Production cluster at UTK.
- Input data and result data are already compressed in Portable Network Graphic format (PNG), so no additional compression is performed during data transfer operations.

Globus Benchmark Results

Globus Timing Breakdown

<table>
<thead>
<tr>
<th>Input Size</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.25MB</td>
<td>50</td>
</tr>
<tr>
<td>1.45MB</td>
<td>40</td>
</tr>
<tr>
<td>718KB</td>
<td>30</td>
</tr>
<tr>
<td>414KB</td>
<td>20</td>
</tr>
<tr>
<td>104KB</td>
<td>10</td>
</tr>
</tbody>
</table>

Input Size: 2.25MB, 1.45MB, 718KB, 414KB, 104KB

- Time (s): Retrieval, Compute, Overhead, Staging
**NetSolve Benchmark Results**

NetSolve Timing Breakdown

- **Time (s)**: Y-axis
- **Input Size**: Bars for different data sizes: 2.25MB, 1.45MB, 718KB, 414KB, 104KB
- **NetSolve** vs **Globus**

**NetSolve / Globus Results Comparison**

Average Total Run Time

- **Time (s)**: Y-axis
- **Input Size**: Bars for different data sizes: 2.25MB, 1.45MB, 718KB, 414KB, 104KB
- **NetSolve** vs **Globus**
NetSolve / Globus Results Comparison

**Average Data Staging Times**

- **Input Size**:
  - 2.25MB
  - 1.45MB
  - 718KB
  - 414KB
  - 104KB

- **Time (s)**
  - 0.00
  - 0.10
  - 0.20
  - 0.30
  - 0.40
  - 0.50
  - 0.60
  - 0.70
  - 0.80
  - 0.90
  - 1.00
  - 2.25MB
  - 1.45MB
  - 718KB
  - 414KB
  - 104KB

**Average Overhead**

(Not including data transfers)

- **Input Size**:
  - 2.25MB
  - 1.45MB
  - 718KB
  - 414KB
  - 104KB

- **Time (s)**
  - 0.00
  - 2.00
  - 4.00
  - 6.00
  - 8.00
  - 10.00
  - 12.00
  - 14.00
  - 16.00
  - 18.00
  - 2.25MB
  - 1.45MB
  - 718KB
  - 414KB
  - 104KB
Conclusions

- For this application the NetSolve implementation is superior to the Globus implementation.
- Globus carries a larger overhead than NetSolve.
- The degree of variability in the Globus measurements is much greater than the variability of the NetSolve measurements.
- Fluctuations in network traffic and server load may have influenced the large increase in variability in some of the test cases.
- Globus may have better results when used over a WAN or with larger data files, due to its ability to use multiple parallel data streams during data transfers.

Lessons Learned

- Grid magnifies performance-related problems we haven’t solved well on large scale systems, SMP, or in some cases sequential processors.
- Performance evaluation is hard
  - Dynamic nature
- Automate the selection
  - User doesn’t want or know how
- Need performance model
  - Automagic would be best
- Need info on grid performance (NWS)
  - BW/Lat/processor/memory
- Monitoring tools
- Performance diagnostic tools are desperately needed.
  - Lack of tools is hampering development today.
- This is a time for experimentation, not standards
Conclusions: What is Needed

- Execution infrastructure for adaptive execution
  - 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
  - Problem-solving environments
- Robust reliable numerical and data-structure libraries
  - Predictability and robustness of accuracy and performance
  - Reproducibility and fault tolerance
  - Dynamic reconfigurability of the application

Thanks to

- Sudesh Agrawal
- Brian Drum
- Shirley Moore
- Keith Seymour
- Zhiao Shi
- Asim YarKhan