GLOBAL 3D SEISMIC TOMOGRAPHY USING MULTI-CORE DISTRIBUTED MEMORY PARALLEL COMPUTERS

James R. Hipp, Marcus C. Chang, Mark A. Gonzales, Benjamin J. Lawry, Andre V. Encarnacao, and Glenn T. Barker, *Sandia National Laboratories*

SAND2009-5843C

Sandia National Laboratories

National Nuclear Security Administration

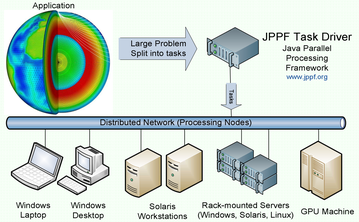
Introduction

The major impetus for this work is to develop a platform-portable, cost-effective, scalable, and high performance parallel software system for solving 3D computationally intensive Ground-based Nuclear Explosion Monitoring Research & Development (GNEMRD) problems within a reasonable time frame (minutes-hours). Our system is comprised of an arbitrary number of affordable modern commodity, multi-core computers linked together in a parallel fashion using a simple two component framework. These primary components include a processing node resource management system that allocates physical processing nodes, memory, and I/O capabilities to one or more requesting applications, and a task distribution system that handles computational work unit assignment for each application. We use this system to perform a number of computationally intensive geophysical problems including ray-path prediction, ray similarity/clustering computations, 3D regional and world Earth tomography calculations, event location, and waveform correlation event detection processing. Our eventual goal is to refine this software system into a deployable package that can be easily installed at other sites where GNEMRD research products are developed to improve the overall U.S. monitoring capability.

In this report we describe the current system developed at Sandia including the two primary components mentioned above. We also describe the basic process of how parallel tasks are solved using this system and how different problem types are categorized into coarse- and fine-grained solution methodologies.

Parallel Process (How the system works)

The basic computational problem is split into many sub-problems, or tasks, and calculated independently on one of the many processing nodes comprising the system.



Performing this process efficiently involves three interacting subsystems.

- The Client Application (CA) – where the code starts (main()), processes results, and ends.
- The Node Resource Manager (NRM) – which monitors system health, security, and the status of requesting CAs. The NRM is responsible for allocating and assigning processing node resources (CPUs and memory) to CAs.
- The parallel task distribution system – We use the Java Parallel Processing Framework (JPPF, **www.jppf.org**) for this component. This system assigns the tasks provided by the CA to the compute nodes assigned by the NRM. The JPPF system is composed of three primary sub-components including the ***Client***, which resides with the CA and acts as the CAs interface with JPPF; the ***Driver***, which lives on a machine prescribed by the NRM and handles task distribution to the third component; the ***Nodes***, which processes each task and returns the results to the ***Driver***.

The parallel process occurs over three phases: Startup, Operation, and Shutdown. Each phase is shown to the right.

Startup:

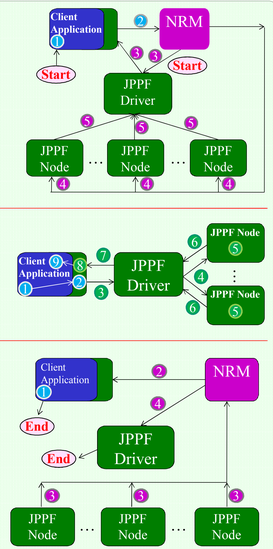
1. Client Application (CA) starts up.
2. CA requests processing nodes from NRM.
3. NRM starts JPPF Driver and assigns to CA.
4. NRM accumulates JPPF Nodes.
5. NRM assigns JPPF Nodes to JPPF Driver.

Operation:

1. Meanwhile, CA creates and assembles list of parallel tasks.
2. CA submits list of parallel tasks to resident JPPF client.
3. JPPF Client sends task list to JPPF Driver (over network).
4. JPPF Driver distributes tasks across network to JPPF Nodes.
5. JPPF Nodes execute tasks.
6. Upon completion, JPPF Nodes send results back to JPPF Driver.
7. JPPF Driver forwards results back to JPPF Client.
8. JPPF Client notifies CA of the arrival of new results.
9. CA accumulates and processes the results.

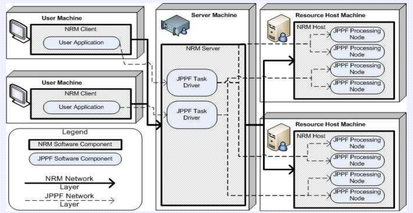
Shutdown:

1. Ultimately, CA finishes work and shuts down.
2. NRM senses CA has shut down.
3. NRM releases JPPF Nodes from Driver.
4. NRM shuts down JPPF Driver.



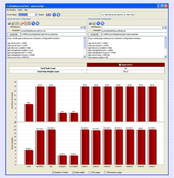
Node Resource Manager (NRM)

- Developed by Sandia to perform automated resource management including starting and stopping processing nodes and sharing node resources among multiple simultaneously executing applications.
- Handles multiple Client Applications (CA) requesting simultaneous access to processing nodes.
- Configures, starts, and stops processing nodes.
- Starts and stops JPPF Drivers and assigns their processing nodes.
- Performs node balancing operations among competing CAs.
- Allows users to contribute their desktop systems as resource hosts (processing node host).
- Robust automated failure recovery (including NRM, JPPF Driver, and processing node).
- Platform independent (Windows, Linux, Unix, and Apple operating systems) and both 32- and 64-bit compatible.



API/GUI

- Easy-to-use API for connecting CAs to the distributed system.
- Contains many features including:
  - Start/stop JPPF Drivers.
  - Request/release JPPF Nodes.
  - Configuration management.
  - CA processing node allocation
  - Various monitoring capabilities.



Host Installer

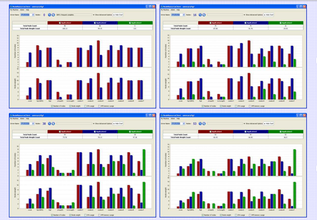
- Uses a database to identify and track what hosts (and associated resources) are available for use by the system.
- Facilitates end-user resource sharing by allowing their own computing resources to be connected to system.
- Defines a policy based approach for defining host usage configuration.

Application node balancing

- Fairly shares processing nodes among multiple users in a transparent fashion.
- NRM maintains a weighted node count metric based on:
  - Application priority.
  - Performance weight for each node used by the application.
- Certain events can unbalance system.
- Dynamic reassignment is used to rebalance system.

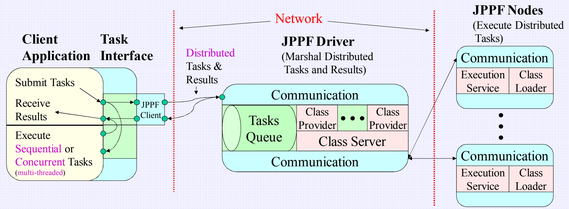
Example illustrating node balancing when a new application starts up.

a) Application 3 (green) at initial startup has no nodes (160 nodes shared evenly by applications 1 (red) and 2 (blue)).
b) Application 3 at ~20 seconds from startup has 16 assigned nodes.
c) 38 assigned nodes after ~ 40 seconds.
d) Balance achieved ~ 1 minute after startup.



Java Parallel Processing Framework (JPPF www.jppf.org)

- Java based platform-independent framework.
- Sandia designed front-end interface to support "sequential" and "concurrent" (multi-threaded) modes.
- Allows a single machine to process in multi-threaded mode or sequentially (helpful for debugging).
- JPPF Client is instantiated by the Client Application (CA) and passes submitted tasks to the JPPF Driver.
- JPPF Driver distributes tasks among JPPF Nodes as they become free (ready for a new task).
- JPPF Nodes execute tasks in parallel. Task written by user to perform necessary parallel calculation.
- JPPF can be executed in Data-Parallel manner with information shared between nodes.



Coarse-Grained/Fine-Grained Parallelism

- Computational parallelism spans a broad range of granularity from very coarse-grained (e.g. ray-prediction, ray similarity and clustering) to extremely fine-grained (e.g. LSQR, Cholesky decomposition) solutions.
- Coarse-grained parallelism is often referred to as "Task" parallelism.
- Fine-grained parallelism is often referred to as "Data" parallelism.

Multi-core CPU versus GPU:

- Multi-core CPU (Central Processing Unit) … Optimally used for "Task" parallel problems.
  - Traditional computer processor used in everything from laptops to servers.
  - Each processor can contain multiple processing cores (typically up to 4 or 6).
- Multi-core GPU (Graphics Processing Unit) … Optimally used for "Data" parallel problems.
  - Traditionally viewed as part of the video card, used by computers to render graphics. Can now be exploited to perform mathematical computations.
  - Program using Nvidia's CUDA language (C-based extension).
  - Nvidia Tesla Card contains 240 cores per GPU, with up to 4 GPUs in a single machine (960 total cores).
- A single machine can contain multiple CPUs and GPUs.

Task Parallelism	Data Parallelism
Distributes execution *tasks* across multiple processing nodes	Distributes *data* across multiple processing nodes
Possesses large flop count / memory access ratio (>> 1)	Possesses small flop count / memory access ratio (≤ 1)
Each node may execute a different task/ operation	Every node executes the same operation over a different range of the data
Requires defining tasks	Requires breaking down the data into subsets that can be operated on in parallel
Typically has no inter-node communication requirements	May require significant inter-node communication
Optimal for "embarrassingly parallel" problems that are easily formulated into independent tasks	Optimal for large data problems that undergo a small set of repeatable operations
Example: Predicting independent ray paths through an Earth model	Example: Solving a large (multi-GB) system of simultaneous equations

Conclusions and Future Direction

Development of this distributed parallel processing framework has reduced the computational time required to solve complex three dimensional geophysical problems at Sandia from months to hours. If 3D calculations are to remain practical and cost-effective then they will only be so if multi-core and GPU hardware are used extensively in their solution. In the future we plan on exploring the GPU as a resource for speeding up fine-grained (data) parallel problems.

Sandia is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.