Running Parallel Simulations and Enabling Science Gateways with the NSF MATLAB Experimental Computing Resource at Cornell

Steven Clark, Brandon Hill and Pascal Meunier
SC10 Tutorial, November 2010
Presentation Overview

• The nanoHUB Science Gateway
• Project Objectives
• nanoHUB to CAC framework concept
• Security aspects
• Overview of tools converted to use CAC
• Inner Mechanisms of the Framework
• Demo of a tool parallelization (pi)
Science Gateway

- Definition from TeraGrid.org:
  - Community-developed set
    » Tools
    » Applications
    » Data
  - Integrated via a portal or a suite of applications
  - Customized graphical user interface
Science Gateway Users

• Categories
  » Students doing coursework
  » Autodidacts
  » Researchers
  » Tool authors

• Resource management must be transparent to users

• Authors adapting their tools to the gateway must have as little work and learning to do as possible

• User interface must be easy to use
Objectives for this Project

- Given a MATLAB tool that is already running on the science gateway:
  - Minimize the number of code changes required to make it use CAC
    - Automation of compilation
    - Abstraction and generalization of support code
  - Minimize the learning curve for the author
- Only considering independent parallel runs at this point
  - No communication needed between tasks
• For new projects:
  » Provide a viable alternative to using C/C++, Fortran (etc...) to other parallel frameworks (e.g., MPI, Message Passing Interface)
    – Problem: hardware-specific components make them non-transportable
    – Must be built wherever they are run
    – Performance depends on expertise
  » MATLAB can be compiled once and run elsewhere
    ✓ Less expertise required
    ✓ Many science students are proficient in MATLAB
Cyberinfrastructure for Running Tools

- Maxwell's Daemon Middleware
- Content Database
- Physical Machine
- Virtual Container
- Simulation
- Tool session cluster
- Rendering Farm
- nanoVIS
- submit
- results

Rendering Farm
Maxwell’s Daemon Middleware
Physical Machine
Virtual Container
Simulation
Tool session cluster
Rendering Farm
nanoVIS
submit
results
General Case

• Interactions between nanoHUB tools and CAC involve:
  » Managing our CAC account space
  » Creating the job and associated tasks
  » Managing the inputs and outputs of each task when it runs
  » Retrieving the results
  » Post-processing on nanoHUB
Prerequisite

• Installation of the CAC client software
  ✓ Library for MATLAB to talk to CAC
  ✓ Change paths and settings in the MATLAB installation to make use of the library
  • Files to be changed:
    – classpath.txt
    – pathdef.m
    » in matlab_root and mcr_root
  • Only needs doing once
  • Instructions available from CAC
  ✓ Transparent to the tool authors
Prototype Framework Directory Structure

• Structure helps understanding and focus
• A tool directory has the following sub-directories:
  » submitCAC
    ✓ Author ignores what's here
  » remote_app
    ✓ Existing tool code goes here
  » retrieveCAC
    ✓ Author ignores what's here
  » post-processing
    ✓ New code goes here
What the Author Needs to Do

• Author edits a “tool_constants” file
  ✓ Tool name
  ✓ Names of files to retrieve
  ✓ Input string (e.g., name of an input file)
  ✓ MATLAB version to use

• Author puts tool code in the remote_app directory

• New code to:
  » Split the work into N tasks
    ✓ Generate task parameters and data for each
  » Assemble the results
Approach

• Tool code often has loops
• Identify the loops
• Split them into smaller ones
• Using this prototype framework, parallelization and adaptation to run on CAC is a very small fraction of the effort needed to develop a tool
  » but no metrics yet
  ✓ Authors often don't know how much time they really spent developing a tool anyway
Managing the CAC Environment

• Isolate tools
  » A directory for each tool source
  » A directory for all the jobs of a tool
    ✓ Failed runs can leave files lying around, it can get messy after a while

• Isolate Jobs
  » A directory for each invocation of the tool, with a unique identifier

• Isolate Tasks
  » A directory for input and output files for each task in a job
Securing the CAC Environment

• CAC provides us with accounts for:
  » Production
  » Development
    ✓ By staff
    ✓ Goal: we want to extend access to MATLAB development using CAC by vetted users, who also have a valid license
      – If we “compile” their code then it can run elsewhere
        ➤ This satisfies the license terms
        ➤ We send .m files that have been encrypted during compilation
Production Security

• All nanoHUB production jobs are coalesced and run as a single CAC user

• The code can't be changed by regular users
  » Staff intervention is required to put new code in production
    ✓ Difficult for staff to find hidden malicious code, but the possibility of inspection is a deterrent
  » Even tool authors can't directly change production code
  » Difficult to erase evidence of malicious code in the tool after an attack
    ✓ Accountability
Security During Development

• Same development user operating on behalf of all tool authors

• Risk: shared development environments may enable competing teams to interfere with the other's work
  » They can run arbitrary code within the shared account!

• Code can be changed on a whim
  » Deniability
  » MATLAB can read (or modify) arbitrary files from different tool authors
  » MATLAB can execute arbitrary programs ("dos" command)

• Attacks between tool authors, or on the infrastructure, are possible
Development Security

• Code inspection by staff is impractical
• We currently vet code authors and restrict access
• Additional possibilities:
  » One development account per tool
    ✓ Would require a pool of development accounts given to nanoHUB
  » Or, tool authors should not use CAC until tool development no longer requires confidentiality
    ✓ Develop new features separately and port when done
Code Using CAC

- Probe
  - Measure CAC availability & run a few simple commands
- NanoNet
- NanoMOS
- Pi Demo today
  - Calculates pi
    - Method: Integrals
- More to come
Probe CAC Availability

- One dot every time the probe runs
  - Starts 30 min after previous one ends
NanoNet

• Monte Carlo simulation
  ✓ random number generator
  ✓ repeat random runs
  ✓ Also with sweeps

![Graph showing Drain Current vs. Channel Length with different colors for different D values.]

Current = 0.77219
NanoMOS

• Space partitioning
• 3-D surface that is described by plots of 2-D curves
• Parallelize computation of each curve
• One core/curve, ~10 curves
Inner Mechanisms of the Framework

- Activities
  - Compilation
  - Running
- Deployment on nanoHUB machines, accounts used
• Various MATLAB scripts are initially in the form of templates
  » Automatically customized for a tool

• Pre-processing of templates
  » “Magic strings” are replaced with actual values derived from constants (e.g., tool name)

• Makefiles are chained in hierarchical fashion
  » Assemble MATLAB scripts from libraries and templates
  » Use functions to decrease code duplication
• CAC account space is managed for the tool, creating the needed directories
  » Make sure no old code is still present

• MATLAB scripts:
  » Compiled
  » Take encrypted .m files
    ✓ Extract
    ✓ Transfer to CAC (for those that need to be)

• Manager Python script also created
Running the Framework

- Manager oversees the sequence of steps

![Diagram showing Manager overseeing cac_submit.m and cac_retrieve.m](image_url)
Manager

• Activity Diagram

- Verify Certificate
- Create UNIX environment
- Set Signal Handlers
- Invoke cac_submit.m
- Cancel Job
- N Tasks
- Wait for task completion
- cac_retrieve.m
- Cleanup

X
Deployment

- Gateway User ID
  - Example: uhub

- Submission Host
  - Proxy
  - Submission user
  - Example: usubmit

- Container
- NFS

- CAC User
  - Example: nanodev3

- CAC

(Check submit permissions, manage certs, Internet proxy)
Experience

• Overhead in compiling and running tools on CAC suggests a more cautious development cycle

• M-Lint sometimes saves time by finding errors faster
  » In MATLAB, open the source file and go to Tools => M-Lint => Show Report
  » However, M-Lint misses things that other code scanners catch
    ✓ e.g., PyLint for Python code will catch variables used before they are assigned a value
• Questions