High-Performance Scientific Computing for Astrophysics

Astro 585, Spring 2014

Class Projects

The class project (worth a total of 50% of final grade) includes the following key elements:

- a written proposal outlining your project (5%),
- implementing a solution to your problem that passes your tests and uses programming practices from class in time for the peer code review (10%),
- performing a helpful code review on a peer's project (5%),
- optimizing performance for a multi-core shared-memory system (i.e., modern workstation; 10%),
- optimizing performance using either a distributed memory system (e.g., RCC cluster), a many-core accelerator (e.g., GPU or Intel Phi), or on the cloud (e.g., Amazon Elastic Compute Cloud, MultyVac?) (10%), and
- a 15 minute presentation to the class describing your project, documenting the performance of different versions of your code as a function of problem size and lessons learned (10%).

Project Proposals

I've provided a few example ideas that could become the basis for a class project (see page 3). You may choose one of these or you may develop an independent proposal more closely related to your research interests. If you choose to develop an independent proposal, then you should discuss it with me far enough in advance of the deadline, that you can refine or change your plans prior to the proposal due date (Friday, February 21). In either case, the written proposal should include the project goal, a description of the inputs, a description of the outputs, a plan for how you will test your code, a discussion of the relevant range of problem sizes, what computer architectures, programming languages, libraries and parallel programming patterns you will use, as well as an explanation of your choices. See the grading rubric (page 2) for a list of issues to think about while developing your plans and questions to address in your proposal.

Grading Rubric for Class Project Proposals (v1.0)

1. Project Summary *(1 point)*

What are the project goals (i.e., overall purpose of project, potentially extending beyond the duration of the class project)?

What are the project objectives (i.e., specific tasks that you will implement, optimize and parallelize)?

How will the objectives contribute to the project goal?

What will be the input parameters and/or data files for your code?

What will be the outputs of your code?

Will you work alone or collaboratively? If you will work as a pair or team, describe your role and your expectations for other team members.

2. Initial Implementation Plan *(1 point)*

What do you anticipate will likely be the most time consuming portions?

How run time will scale with problem size?

What is the relevant range of problem sizes? (There should be at least one size parameter that you will vary when benchmarking.)

What programming language(s) will you use? If not Julia, then provide a justification of why your choice makes sense for your project.

What libraries or existing project will you use? What you will use them for? If there are multiple possible libraries for some functionality, then explain your choice.

If you will be working collaboratively, how will team members coordinate their effort? What internal deadlines must be met for all team members to complete project on time?

3. Testing Plan *(1 point)*

What unit tests will you use?

What will you use for regression tests?

4. Parallelization Plan *(1 point)*

What aspects of the calculations will you parallelize?

What computers architectures will you use for parallelizing your code (e.g., multi-core workstation and a cluster of nodes)? Why is this a good choice for your problem?

What parallel programming tools/patterns you will use (e.g., parallel for loops, parallel map/mapreduce, OpenMP, MPI, CUDA, Thrust)?

What specific machines will you use? (e.g., my Windows laptop for initial development and testing with a single core, my six-core linux workstation and 16 cores on Lion-XH cluster)

Are there any additional libraries that you plan to use for the optimized and/or parallelized versions of your code? If so, what you will use them for? If there are multiple possible libraries for some functionality, then explain your choice.

5. Overall Strength of Proposal *(1 point)*

Is the proposed project and relevant data size well-suited to parallelization?

Are the proposed tasks realistic, given the student's/team's current level of experience?

Example Goals for Class Projects

(I've listed a high-level overview of the steps that would be required to help you choose a project and get started. Even if you pick one of these, you'll still need to think through the problem yourself (or as part of a small team) to develop a full project proposal that addresses each of the issues listed in grading rubric.)

1. Search for Transiting Planet Candidates in Kepler data

    Load Light Curves from FITS files (see kplr package)

    Strip out bad data (i.e., quality flag!=0, sap_flux or pdf_flux = NaNs, Infs, first day of each quarter, etc.)

    Strip out data points near known transits candidates

    Identify usable light curve segments (i.e., define segment breaks whenever several bad points in a row, require segments to have at least a minimum number of points)

    Normalize each segment (e.g., median value or low-order polynomial)

    Estimate noise level for each quarter/month/segment (e.g., median absolute deviation, clipped rms)

    Apply search algorithm (e.g., Box Least Square; see Kovacs et al. 2002; Aigran & Irwin 2004)

    Find period and epoch that maximize signal-to-noise

    If statistically significant, loop and repeat until no more statistically significant signals

2. Compute Sensitivity for Planet Detection (as function of planet size, period)

    Compute simulated transit light curve models

    Injecting fake transit signals into real light curves

    Apply Search for Transiting Planet Candidates above to simulated data

3. Fit Transiting Planet Models

    Load Light Curves from FITS files (see kplr package)

    Strip out bad data (i.e., quality flag!=0, sap_flux or pdf_flux = NaNs, Infs, first day of each quarter, etc.)

    Strip out data points near known transits candidates

    Identify usable light curve segments (i.e., define segment breaks whenever several bad points in a row, require segments to have at least a minimum number of points)

    Normalize each segment (e.g., median value or low-order polynomial)

    Estimate noise level for each quarter/month/segment (e.g., median absolute deviation, clipped rms)

    Start w/ initial guess for transit light curve parameters (e.g., near published parameters for planet candidates)

    Compute log likelihood for assumed transit light curve parameters

    Repeat while adjusting light curve parameters (e.g., Levenberg-Marquart algorithm to maximize likelihood for best-fit model; Markov chain Monte Carlo to estimate parameter uncertainties

4. Search for Transit Timing Variations (TTVs)

    Fit a Transiting Planet Model that assumes constant spacing between transits

    Refit a more general model that allows for simple parametric form for TTVs (e.g., quadratic or sinusoid), while holding transit shape parameters fixed.

    Refit holding transit times fixed, while allowing shape parameter to vary again

    Refit the more general model while holding transit shape parameters fixed.

    Compute log likelihood for generalized transit light curve parameters

Repeat while adjusting transit emphemeris parameters (e.g., Levenberg-Marquart algorithm to maximize likelihood for best-fit model; Markov chain Monte Carlo to estimate parameter uncertainties