

DRAFT

Educational Competencies for Petascale Computing

Ohio Supercomputer Center
May 2011

Please send any comments to Steve Gordon (sgordon@osc.edu)

The Ohio Supercomputer Center is working under a contract from TeraGrid to help define the educational competencies associated with undertaking the use of leadership-class computing (petascale and beyond) in science and engineering research. The intent is to draft competencies and lead discussions across the high performance computing community to come to a consensus on the topics that should be incorporated into graduate programs so that a new generation of researchers is prepared to take advantage of the extraordinary computational power at the leadership level.

Below is a set of competencies based on multiple surveys and discussion panels. An initial survey of faculty, graduate students, industry, and government labs was completed in spring 2009. The respondents provided their rankings of the skills required for undertaking Petascale computing and becoming part of the research community applying those skills to science and engineering research. Based on those results we developed an initial draft of required competencies. The draft was presented at the Teragrid 2010 conference, and feedback was solicited through an online survey. There were 88 responses to the survey by a variety of faculty, student, and researchers. The survey showed broad agreement on many of the competencies but also identified areas where the respondents felt that only a portion of graduate students needed a depth of knowledge.

A summary of the survey results was presented to several small group panels of community leaders through a live, webinar format. Their input was sought on those competencies that should be required for all students and those that would apply only to specialized subgroups of students. Their opinions were discussed at length. Although there was not unanimous agreement, we used these discussions to draft a revised set of competencies that address the basic skills that the majority of participants agreed are important for all students.

This document presents a revised set of competencies based on the feedback we received. The following groups of competencies are not intended to represent individual courses. Competencies from different categories will most likely be combined to form courses as academic programs are created or modified.

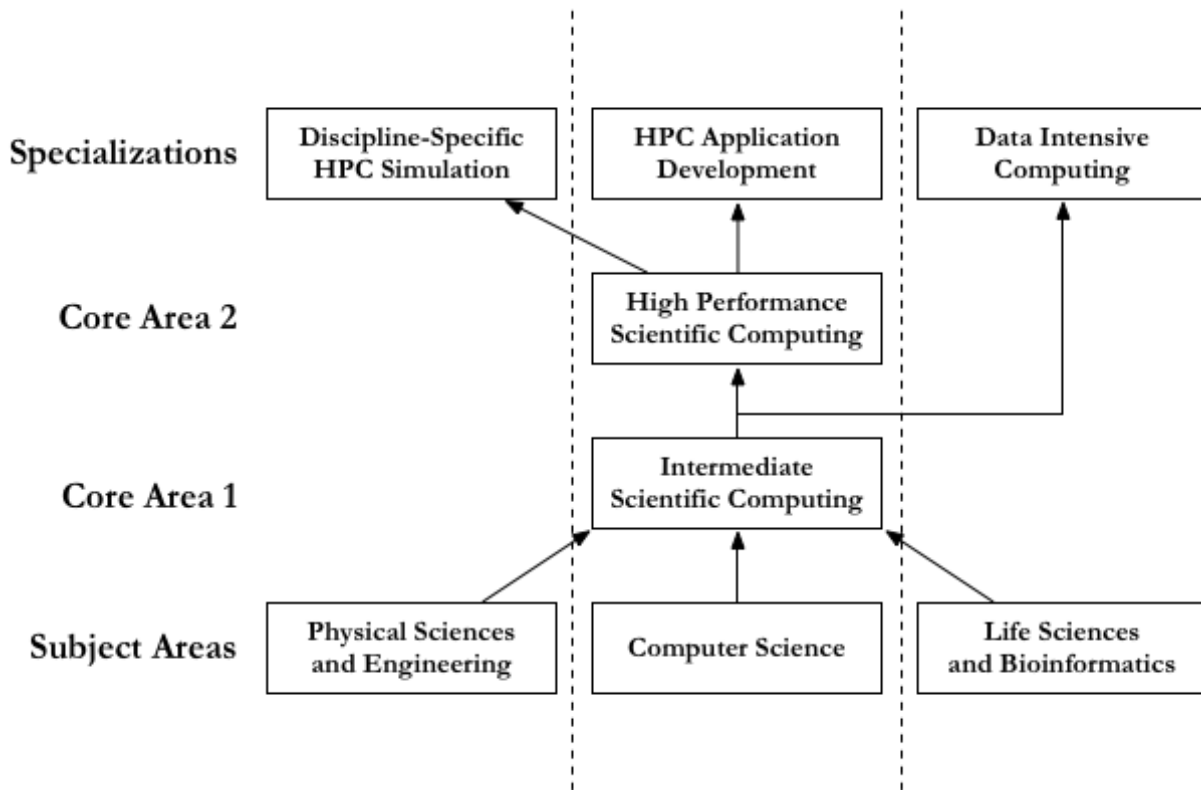
The skills are split into two levels – introductory competencies that all students should have and advanced competencies associated with certain subfields. The introductory competencies will most likely comprise two major subareas and have been tentatively split with this in mind. Competencies have been defined for a specialty area in HPC software development. Other specialty areas have yet to be defined but will undoubtedly emerge in further discussions.

The intent of the introductory competencies is to provide all students with an overview of all of the major code development and optimization techniques so that they understand the full range of issues that may arise, can begin to analyze the codes being used in their science or engineering discipline, and can follow best practices for code development so that their work effectively contributes to the advancement of the science.

In the tables of competencies for the different areas, each competency is marked with one of the following codes to indicate the depth of understanding required by the students in the area:

- I – Introduce the topic
- R – Reinforce the topic with applications
- M – Students should master the topic

The diagram below shows the relationships among the various competency areas:



Prerequisites

The specification of the competencies assumes that students have come from undergraduate backgrounds with introductory computational science skills. These could come from a formal program such as the minor program in computational science at the Ralph Regula School of Computational

Science (RRSCS) collaborative institutions (see <http://www.rrscs.org/minor/index.shtml>). This and other related programs generally require the following courses for undergraduates:

- Introduction to modeling and simulation
- An introductory programming class
- Numerical methods
- A discipline oriented modeling and simulation course (Computational X – biology, chemistry, engineering, environmental science, physics, etc.)
- Electives in parallel computing and/or scientific visualization

Competencies for the RRSCS curriculum are shown at <http://www.rrscs.org/minor/competencyfinal.pdf>. We are assuming that students interested in petascale computing will start with these skills or acquire them before taking courses to enhance their skills with the competencies in this document.

Area 1: Intermediate-Level Scientific Computing

Prereqs: Introduction to modeling and simulation, introductory programming (any language), numerical methods, one discipline-oriented computational science course

This course will use either Fortran or C/C++ and will emphasize topics commonly encountered in scientific computing/computational science. It primarily addresses serial computing competencies and is a prerequisite to the HPC/parallel computing area.

This course includes programming skills at an intermediate level and focuses specifically on scientific computing skills.

I/R/M	Competency/Descriptors
R	<p>Ability to contribute code in the programming language of greatest importance in the research domain, either Fortran or C</p> <p>Descriptors:</p> <p>Students will understand basic language features and concepts</p> <p>Students will be able to understand code written by others</p> <p>Students will be able to use input/output functions effectively</p> <p>Students will complete a project to implement a new code and add a new capability to an existing code</p> <p>Students will be introduced to user-defined types, structures, classes, or similar mechanisms in their primary HPC programming language.</p> <p>Students will be introduced to the concepts of object-oriented design and programming.</p> <p>Example Activity: Students will write a few small programs using common sort and search algorithms.</p>
R	<p>Students will be introduced to basic debugging techniques</p> <p>Descriptors:</p> <p>Students will recognize common runtime errors and program failures and will be able to describe some common mistakes that could cause them.</p> <p>Students will be introduced to debugging strategies and rules of thumb</p> <p>Students will understand capabilities of commonly available symbolic debugging tools such as</p>

	<p>gdb.</p> <p>Students will understand some common memory problems and how to fix them, for example, array overruns, invalid pointers, mismatched parameters in function calls.</p> <p>Example Activity: Students will follow a script to start a debugging session using a symbolic debugger on a sample code containing one or more common errors. The script will demonstrate useful debugging features such as setting break or watch points and inspecting variable values.</p>
M	<p>Understand number representation</p> <p>Descriptors:</p> <p>Students will understand the uses of floating point and integer arithmetic</p> <p>Students will understand the range limits of different size integers and will know how to select the appropriate size.</p> <p>Students will understand the range, precision, and memory requirements of single and double precision floating point numbers and will know how to select the appropriate precision.</p> <p>Students will be able to explain the concepts of floating point representation, including range, precision, overflow, and underflow.</p> <p>Example Activity: Students will do some simple exercises to illustrate the differences among the data types, including allocating large arrays and checking their sizes; experimentally finding the largest representable number (approximately); and for floating point numbers experimentally finding both the smallest representable positive number (approximately) and the machine precision.</p>
R	<p>Understand numerical errors</p> <p>Descriptors:</p> <p>Students will be able to describe various kinds of numerical errors (roundoff, overflow, underflow).</p> <p>Students will be able to describe absolute and relative error.</p> <p>Students will discuss error propagation.</p> <p>Students will understand loss of significance and methods to avoid loss of significance.</p> <p>Students will be able to describe the effect of problem conditioning or sensitivity on the correctness of a computed solution.</p> <p>Example Activity: Students will experiment with solving systems of linear equations involving well-conditioned and ill-conditioned matrices.</p>
I	<p>Students will be introduced to software engineering best practices</p> <p>Descriptors:</p> <p>Students will understand how to define software requirements</p> <p>Students will be introduced to the concepts of software design</p> <p>Students will be introduced to the concepts of unit and regression testing</p> <p>Students will understand the purpose and concepts of source control.</p> <p>Students will use source control for a class project.</p> <p>Students will include useful comments in their code.</p> <p>Students will format code consistently.</p> <p>Students will understand and use the build process.</p> <p>Example Activity: Students will review the software design process with a case study of a computational science code</p>
R	<p>Ability to identify efficient file formats</p> <p>Descriptors:</p> <p>Students will understand the difference between text and binary file formats and the uses and</p>

	<p>limitations of each.</p> <p>Students will be able to estimate output file sizes and storage space requirements for the results of a simulation in their area of interest.</p> <p>Example Activity: Students will do textbook style exercises to estimate storage requirements and data transfer times for storing and moving large files in both text and binary formats.</p>
R	<p>Create a program that uses at least one widely used numerical library</p> <p>Descriptors:</p> <p>Students will demonstrate how to include the appropriate header files.</p> <p>Students will demonstrate how to call the library routines, pass in arguments, and use the output values.</p> <p>Students will demonstrate how to compile, link and run the program.</p> <p>Students will understand the effect of environment variables or other settings on library performance.</p> <p>Students will be able to discuss library quality: differences between excellent and questionable libraries.</p> <p>Example Activity: Students will compare the performance of naively coded matrix multiply to that of a well-crafted library implementation for a variety of matrix sizes.</p>
I	<p>Students will be introduced to the concepts of algorithm complexity</p> <p>Descriptors:</p> <p>Students will understand at the overview level the concept of algorithm complexity and will know how some important algorithms compare to each other.</p> <p>Students will be aware of big-O notation for algorithm complexity.</p> <p>Example Activity: Students will do textbook problems on complexity analysis for common linear algebra and search algorithms.</p>
I	<p>Students will be introduced to serial program optimization concepts and techniques</p> <p>Descriptors:</p> <p>Students will understand basic compiler optimization options and how to use them.</p> <p>Students will experiment with compiler optimization flags to see the effect on performance of selected codes.</p> <p>Students will understand the dangers of using compiler optimizations on floating-point code, where common optimizations can change results.</p> <p>Students will use timing functions or profiling tools to measure the time spent in different sections of their code.</p> <p>Students will understand at an overview level the concepts of efficient cache utilization including unit stride and cache reuse.</p> <p>Students will explore these concepts by examining sample codes that have different memory access patterns and measuring the execution time of the different versions. Sample code may include matrix-matrix multiplication.</p> <p>Example Activity: Students will experiment with simple example codes as outlined above.</p>
R	<p>Students will understand verification and validation principles</p> <p>Descriptors:</p> <p>For this competency, students may use a simple model from their field of study or an example provided by the instructor.</p> <p>Students will develop test cases to verify that their programs correctly implement their models.</p> <p>Students will validate their models using one or more of the following approaches: a) identify simple or analytical problems that can be used to validate a new method; b) make a</p>

	<p>statistical comparison to experimental data; c) use a statistical comparison to previous approaches or other models</p> <p>Example Activity: Students will experiment with a simple model as outlined above and write a brief report explaining their model, approach to verification and validation and the results.</p>
I	<p>Students will be introduced to Monte Carlo methods</p> <p>Descriptors: Students will describe applications of Monte Carlo models with examples. Students will discuss algorithms for Monte Carlo methods.</p> <p>Example Activity: Students will write a simple program to compute pi using Monte Carlo integration.</p>

Area 2: Intro to High Performance Scientific Computing

Prereqs: Intermediate-Level Scientific Computing

This area will use either Fortran or C/C++ and addresses HPC and parallel computing topics. The emphasis is on concepts rather than expert-level parallel programming. The course is intended for all users of HPC systems, not just developers.

Students should understand at a conceptual level the various forms of parallel computing and parallel programming models. This course involves concepts that are to a large degree independent of specific technologies.

I/R/M	Competency/Descriptors
I	<p>Students will be introduced to parallel architectures and execution models.</p> <p>Descriptors: Students will understand the concepts of the Single-Instruction-Multiple-Data (SIMD) execution model. They will understand at a conceptual level the architectures associated with this model (streaming and vector processors, including GPUs). Students will understand the concepts of the Multiple-Instruction-Multiple-Data (MIMD) execution model. They will understand at a conceptual level the architectures associated with this model (multicore processors, clusters). Students will understand the concepts of the Single-Program-Multiple-Data (SPMD) execution model. They will understand that SPMD is a subset of the MIMD model used by MPI, UPC and Co-Array FORTRAN applications.</p> <p>Example Activity: Students will take a brief quiz on this terminology.</p>
I	<p>Students will be introduced to memory models for parallel programming.</p> <p>Descriptors: Students will understand the concepts of shared and distributed memory and their advantages and disadvantages. Students will be aware of the differences in programming approaches for shared and distributed memory models. Students will be introduced to the concept of a global address space with distributed memory.</p> <p>Example Activity: Students will take a brief quiz on this terminology</p>
I	<p>Students will understand the principles of how to match algorithms, applications, and architectures.</p>

	<p>Descriptors: Students will be introduced to the concepts of data parallelism, functional parallelism, and task-level parallelism (embarrassingly parallel problems). Students will understand which types of algorithms parallelize well, at what granularity they are parallelized, and how this relates to different architectures. Students will understand what algorithmic and program constructs work well with each of the execution models introduced (SIMD, MIMD, SPMD). Example Activity: Students will do textbook type problems, in which they will identify appropriate models and target architectures for different types of problems</p>
I	<p>Students will understand the concept of application scalability. Descriptors: Students will be able to define weak scalability and give examples of problems that are weakly scalable. Students will be able to define strong scalability and give examples of problems that are weakly scalable. Students will be aware of the factors that limit scalability. Students will understand the limits of parallelism as described by Amdahl's law. Example Activity: Students will do textbook exercises to compute speedup, serial fraction, etc.</p>
I	<p>Students will understand code performance metrics. Descriptors: Students will know how to measure, interpret, and report the performance of their code. Students will know how to measure, interpret, and report speedup as the number of processors is increased. Example Activity: Students will run a sample parallel program on varying numbers of processors and will calculate speedup.</p>
I	<p>Students will be introduced to parallel programming methods and concepts. Descriptors: Students will be able to identify parallelism in an application and discuss approaches for exploiting it Students will be introduced to the concept of message passing and its implementation using MPI. Students will be introduced to the concept of multithreading and its implementation using OpenMP. Students will be introduced to the concept of vectorization and will understand how to take advantage of it using compiler flags. Students will be introduced to the concepts of streaming and many-core accelerators such as GPUs. Students will be aware of various concurrency issues, the problems they can cause, and some mechanisms for avoiding them, including race conditions, deadlocks, critical sections, and data dependencies. Students will discuss the generation of parallel random number streams. Example Activity: Students will review and run MPI and OpenMP example programs on parallel computer system.</p>
I	<p>Data Intensive Computing Descriptors: Students will be able to define data intensive computing and explain how it differs from traditional high performance computing.</p>

	<p>Students will describe an application, preferably related to their field of study, that involves data intensive computing.</p> <p>Students will be able to transfer a dataset from one system to another using a parallel data transfer method .</p> <p>Students will understand how data intensive computing impacts algorithm design.</p> <p>Example Activity: Students will experiment with file transfer tools such as gridftp.</p>
I	<p>Data management</p> <p>Descriptors:</p> <p>Students will be familiar with the file compression and archiving tools available on their systems.</p> <p>Students will know what tool to use to decompress a file in any of the formats commonly used in the HPC community.</p> <p>Students will understand how metadata helps make sense of data and will recognize at least one metadata format, possibly XML.</p> <p>Example Activity: Students will read and write HDF5 files with a provided application and will use HDF5 commands to inspect the files</p>
I	<p>Understanding fault tolerance</p> <p>Descriptors:</p> <p>Students will be able to explain the concept of mean time between failures (MTBF).</p> <p>Students will be able to explain the concept of fault tolerance.</p> <p>Students will identify some causes of failures in HPC systems.</p> <p>Students will understand why fault tolerance is a bigger issue on extremely large scale systems than on smaller systems.</p> <p>Example Activity: Students will do textbook exercises to compute MTBF of computer system components.</p>
I	<p>Students will understand basic scientific visualization concepts.</p> <p>Descriptors:</p> <p>Students will understand the difference between scientific visualization and information visualization.</p> <p>Students will understand basic techniques to visualize scalar fields.</p> <p>Students will understand basic techniques to visualize vector fields.</p> <p>Example Activity: Students will display a two-dimensional scalar dataset using a widely available visualization tool, for example, MATLAB, Excel, or VisIt.</p>

Specialty Area 1: HPC Software Development

Prereqs: Intro to High Performance Scientific Computing

This area is intended for researchers who develop HPC software for their own use or community use. Students coming out of this course should understand the topics of the first two areas at a mastery level in addition to mastery of the competencies listed here. There is some redundancy in the competencies listed for this area and those above; this area is intended to provide more depth.

This area includes more advanced programming skills, software engineering practices, and parallel programming. Students should understand and have a working knowledge of alternative approaches to parallel programming and how they relate to current and emerging parallel programming models.

I/R/M	Competency/Descriptors
M	<p>Advanced programming skills</p> <p>Descriptors: Students will be able to use advanced language features that support scientific computing. Students will be able to create and use user-defined types, structures, classes, or similar mechanisms in their primary HPC programming language. Students will be familiar with object-oriented design and programming. Students will demonstrate the use of gdb or some other commonly available debugger, including setting breakpoints, stepping, and examining the contents of variables.</p> <p>Example Activity: .</p>
M	<p>Students will understand and follow software engineering best practices.</p> <p>Descriptors: Students will define software requirements for one or more of their class projects. Students will be required to produce a software design for one or more of their class projects. Students will use source control for all their class projects and will understand the capabilities of the source control tool they are using. Students will create a makefile for each of their programs or use some other method for one-step builds. Students will develop unit and regression test cases for their programs. Students will be introduced to testing frameworks.</p> <p>Example Activity: .</p>
R	<p>Students will demonstrate techniques for understanding and interpreting existing code.</p> <p>Descriptors: Students will identify the main data structures in a sample code and the relationships among them. Syntactical structure of the code Students will profile code to see call chain and logic. Students will identify calls to external libraries.</p> <p>Example Activity: .</p>
M	<p>Students will be proficient in serial program optimization.</p> <p>Descriptors: Students will correctly use compiler flags to optimize their code. They will understand the benefits and limitations of the compiler optimizations. Students will understand the dangers of using compiler optimizations on floating-point code, where common optimizations can change results. Students will use timing functions and code profiling tools such as gprof to measure the time spent in different sections of their code. Students will understand the concepts of efficient cache utilization including unit stride and cache reuse.</p> <p>Example Activity: Students will explore these concepts by modifying sample code to have different memory access patterns and measuring the execution time of the different versions. Sample code may include matrix-matrix multiplication.</p>
M	<p>Students will use parallel concepts/algorithms in developing software.</p> <p>Descriptors: Students will complete a class project using each of these models: SIMD, MIMD, SPMD. Students will discuss the architectural concepts associated with each of these models and give</p>

	<p>a current example of each.</p> <p>Students will discuss the algorithmic and program constructs that work well with each.</p> <p>Students will understand which types of algorithms parallelize well, at what granularity they are parallelized, and how this relates to different architectures.</p> <p>Students will understand various concurrency issues, the problems they can cause, and how to handle them properly, including race conditions, deadlocks, critical sections, and data dependencies.</p> <p>Example Activity: .</p>
M	<p>Students will demonstrate skill in application scaling.</p> <p>Descriptors:</p> <p>Students will be able to define weak and strong scalability and give examples of algorithms or applications exhibiting each.</p> <p>Students will be able to identify parallelism in an application and decide on the best approach to exploit it.</p> <p>Students will describe several factors that limit scalability.</p> <p>Students will complete a class project in which they measure speedup of their code and present the information in a meaningful way.</p> <p>Students will explain the limits of parallelism as described by Amdahl's law and give examples</p> <p>Example Activity: .</p>
R	<p>Students will be able to write a parallel program using MPI. (Note: Students should learn either MPI or OpenMP programming in some detail and be introduced to the other.)</p> <p>Descriptors:</p> <p>Students will understand the concepts of message passing and communicators.</p> <p>Students will know how to use point-to-point communication and collective communication functions.</p> <p>Students will be able to explain the semantics of blocking and non-blocking send and receive operations.</p> <p>Students will be able to explain the standard, buffered, synchronous, and ready communication modes.</p> <p>Students will recognize common errors that can cause a program to deadlock and will suggest ways to correct them.</p> <p>Students will understand why a program that works correctly with one MPI implementation may fail when run with another one.</p> <p>Students will be introduced to techniques for communicating non-contiguous data or mixed data types</p> <p>Students will be introduced to virtual topologies in MPI</p> <p>Students will understand how data distribution affects communication loads (data movement).</p> <p>Students will investigate the use of non-blocking and one-sided communication for overlapping computation and communication.</p> <p>Example Activity: .</p>
R	<p>Students will be able to write a program using OpenMP. (Note: Students should learn either MPI or OpenMP programming in some detail and be introduced to the other.)</p> <p>Descriptors:</p> <p>Students will use OpenMP for loop-level parallelization in an example program</p> <p>Students will understand what types of data dependencies inhibit loop parallelization</p> <p>Students will be introduced to additional worksharing constructs in OpenMP including Parallel Regions and Parallel Sections</p>

	<p>Students will understand the concepts of private, shared and reduction variables in OpenMP and will be able to correctly determine when to use each attribute.</p> <p>Students will be introduced to synchronization mechanisms available in OpenMP</p> <p>Students will recognize race conditions and use synchronization to prevent errors</p> <p>Students will understand the relationship between threads and cores and how OpenMP relates to both</p> <p>Students will understand OpenMP scheduling options and will experiment with them in class exercise.</p> <p>Example Activity: .</p>
I	<p>Students will be introduced to hybrid or mixed mode MPI-OpenMP.</p> <p>Descriptors:</p> <p>Students should know why and when it is appropriate to mix MPI and OpenMP.</p> <p>Students will be able to explain the benefits and pitfalls of mixing MPI and OpenMP.</p> <p>Students will create and run a program that combines MPI and OpenMP parallelization techniques.</p> <p>Students will be able to explain the threading models in the MPI standard and the strengths and pitfalls of each.</p> <p>Example Activity: .</p>
I	<p>Students will learn techniques for load balancing.</p> <p>Descriptors:</p> <p>Students will be introduced to data distribution strategies such as blocked partitioning, cyclic partitioning, graph partitioning, etc.</p> <p>Students will understand the effect of data distribution on load balance.</p> <p>Students will know how to measure load imbalance.</p> <p>Students will understand static and dynamic load balancing.</p> <p>Example Activity: .</p>
I	<p>Students will be introduced to frameworks for large-scale parallel code development.</p> <p>Descriptors:</p> <p>Students will demonstrate the use of a code development framework or integrated development environment, preferably one created for high performance computing. Potential candidates include the Eclipse Parallel Tools Platform (PTP), Microsoft Visual Studio and Intel Cluster Studio.</p> <p>Example Activity: .</p>
I	<p>Students will be introduced to many-core programming.</p> <p>Descriptors:</p> <p>Students will explain the difference between many-core and multi-core computing.</p> <p>Students will describe the characteristics, benefits, and programming challenges of at least one many-core device.</p> <p>Students will run a sample CUDA or OpenCL (or similar language) program on a GPU or other many-core device and will compare execution time to that of a CPU version of the same computation.</p> <p>Example Activity: .</p>
I	<p>Students will be able to write a simple Partitioned Global Address Space (PGAS) parallel program.</p> <p>Descriptors:</p> <p>Students will understand the fundamental concepts of PGAS: partitioned global memory, threads, affinity and nonlocal access, collective operations and owner-computes.</p>

	<p>Students will be introduced to a PGAS language or library such as UPC, Fortran, X10, Chapel, SHMEM or Global Arrays.</p> <p>Example Activity: .</p>
I	<p>Students will understand how to implement checkpointing.</p> <p>Descriptors:</p> <p>Students will be able to explain the purpose and concepts of checkpointing.</p> <p>Students will understand the challenges of checkpointing a parallel code.</p> <p>Students will add checkpointing to an existing HPC code.</p> <p>Example Activity: .</p>
R	<p>Students will demonstrate skill in debugging parallel programs.</p> <p>Descriptors:</p> <p>Students will be proficient in the use of a parallel debugger, for example Totalview, DDT, or the debugging capabilities of an IDE they are using.</p> <p>Students will be aware of some common types of bugs in parallel programs and will understand how to find and correct them.</p> <p>Students will understand how concurrency issues outlined earlier (data dependencies, race conditions, deadlocks) manifest as bugs.</p> <p>Example Activity: .</p>
R	<p>Students will be able to improve the efficiency of a program.</p> <p>Descriptors:</p> <p>Students will know how to improve efficiency by improving data locality and memory access patterns.</p> <p>Students will know how to improve efficiency by finding hotspots in their code and optimizing small sections of code.</p> <p>Students will understand the factors that affect the performance of a parallel program.</p> <p>Students will be introduced to techniques for efficient scheduling: detecting load imbalance, appropriately sized tasks, etc.</p> <p>Students will be introduced for techniques for reducing communication overhead such as message coalescing and latency hiding.</p> <p>Example Activity: .</p>
R	<p>Students will know how to use performance analysis tools.</p> <p>Descriptors:</p> <p>Students will understand how to use performance analysis tools such as gprof, IPM and TAU to find performance problems in their code.</p> <p>Students will be able to interpret the output of performance analysis tools.</p> <p>Students will be able to use profiling tools such as Oprofile which report Hardware Performance Counter data to measure CPU performance metrics such as cache misses.</p> <p>Example Activity: .</p>
R	<p>Students will understand HPC workflows and know how to automate them.</p> <p>Descriptors:</p> <p>Students will learn the basics of shell scripting, Perl, Python or some other language used for scripting.</p> <p>Students will create a script to automate a simple HPC workflow.</p> <p>Students will be introduced to workflow construction tools such as Kepler and Eclipse.</p> <p>Example Activity: .</p>
R	<p>Students will understand parallel I/O.</p>

	<p>Descriptors: Students will understand how to implement parallel I/O, for example using MPI I/O, HDF5, or ADIOS Students will create a program that uses MPI-IO or some other library to perform parallel I/O. Students will understand the effect of network topology on parallel file I/O. Students will investigate leveraging parallel filesystems for data-intensive applications</p> <p>Example Activity: .</p>
--	---

Specialty Area 2: Scientific Visualization

Competencies have not yet been defined for this area.

Specialty Areas: Discipline-Specific

Further discussion is needed with a range of domain-specific researchers to define which topics might be required for particular areas of research.