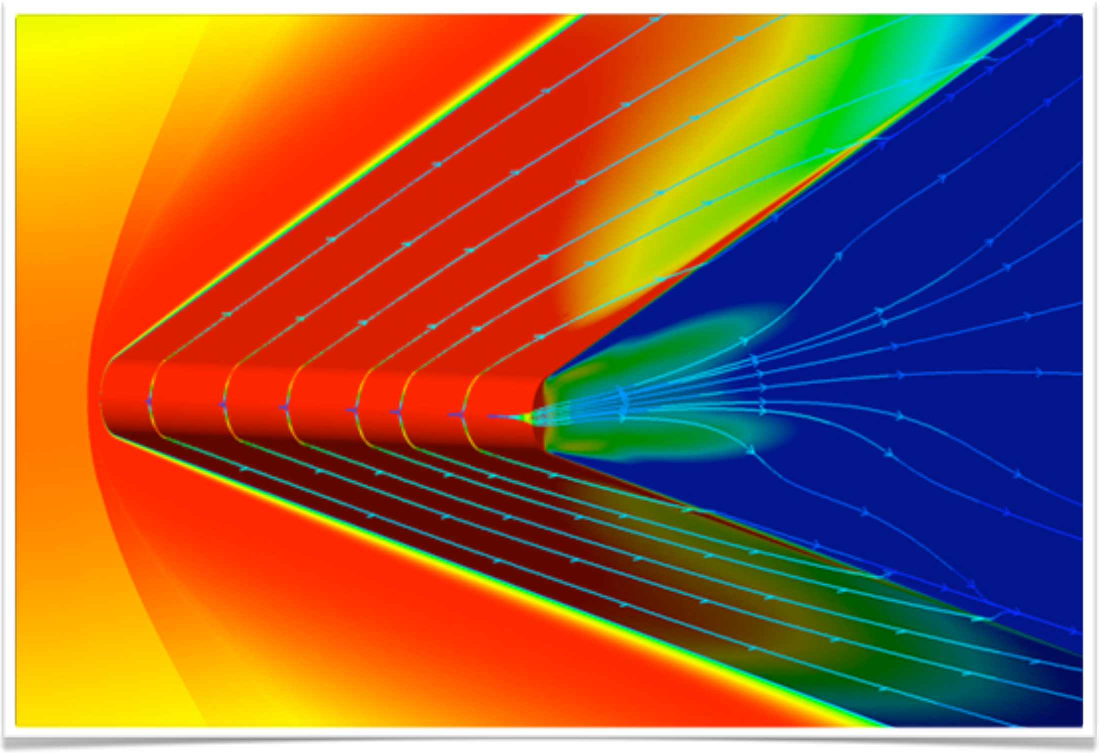
Advanced Cluster Systems

Dauger Research

Supercomputing Engine Technology (SET)

Accelerate Your Software by Parallelizing for Multi-Core Machines or Supercomputers without Source Code Modification



Arc jet simulation of a thermal protection system (TPS) sample wedge in an arc jet flow. Such simulations may use upwards of 180 processors for up to 5 hours, and several of these solutions are often necessary to match the experimental measurements.
Credit: NASA Exploration Systems Mission Directorate, Principle Investigator, Michael J. Wright

Modular Programming

Modular programming is a software design technique where software is composed from separate parts, called modules. Conceptually, modules represent clearly organizing its solution into isolated components, improving the maintainability, and enforcing logical boundaries between components.

Modules are typically assembled into a program using interfaces.
A module interface expresses the elements that are provided and required by the module. The elements defined in the interface are detectable by other modules. The implementation contains the working code that corresponds to the elements declared in the interface.

In module programming, Front End (FE) and Back End (BE) are terms that refer, respectively, to the user-facing and the purely-computational portions of a calculation. The FE is responsible for collecting input in various forms from the user and processing it to conform to a specification the BE can use. The FE is a kind of interface between the user and the BE (The “user” may be a human being or another high-level program.)

Some programs separate the FE and BE into two applications. A FE application is one that users interact with directly. The BE application may interact directly with the FE, but it is possible for the BE to be called by an intermediate program that mediates FE and BE activities.

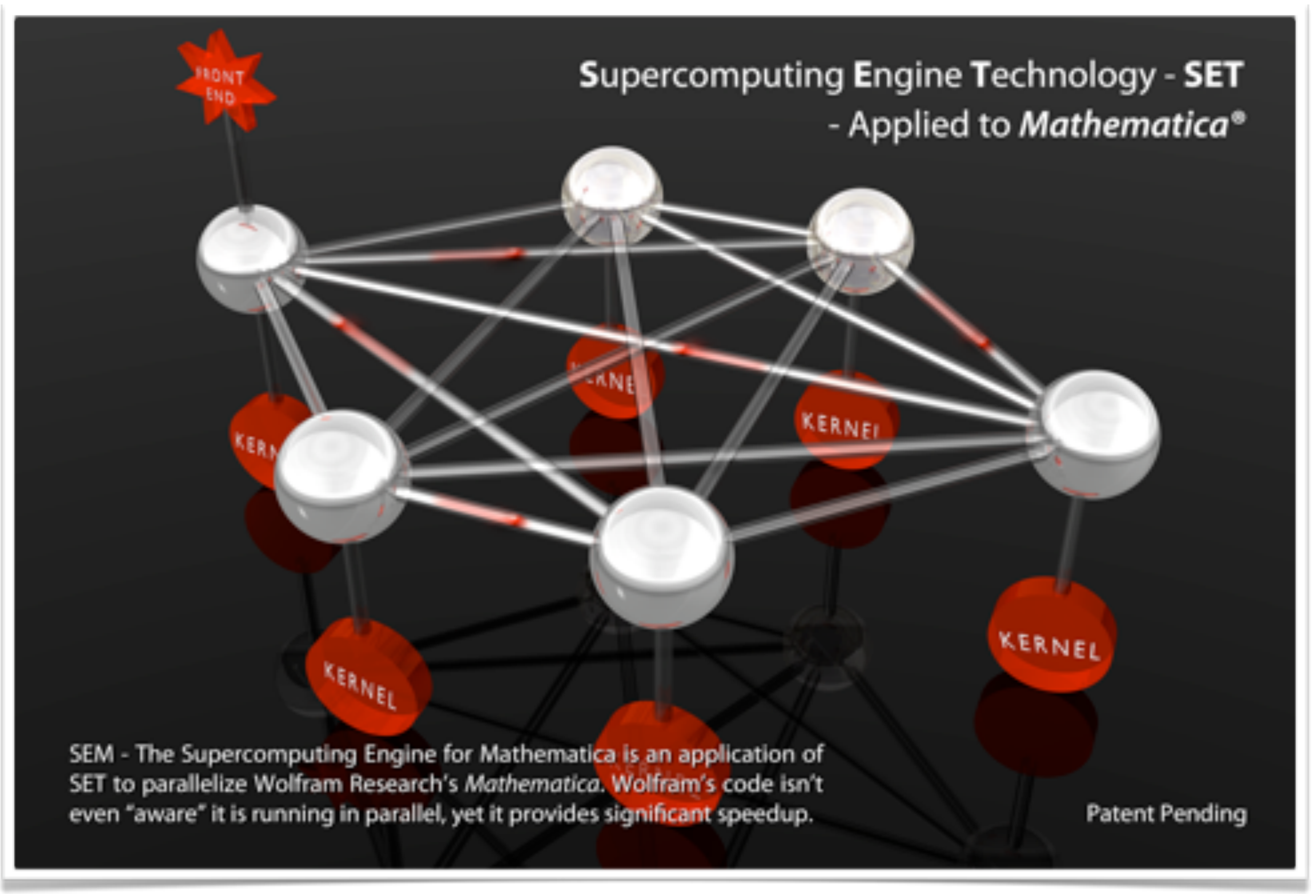
Modular programming is a good practice when writing more reliable application software. It enables faster and easier development of application software, faster and easier testing of the application, and improved maintenance of its operation. Most long-term software projects implement a module programming approach rather than the monolithic approach, where the smallest piece of software is the whole application.

Parallelizing a Modular Program the Standard Way
The design principles of modular programming apply directly to parallel programming. However, parallelism also introduces additional issues. A sequential module encapsulates the code that implements the functions provided by the module’s interface and the data structures accessed by those functions. In parallel programming, not only code and data should be considered but also the tasks created by a module, the way in which data structures are partitioned and mapped to processors, and internal processors communication structures. These processes are quite complex and often involve re-writing large part of the original code.

Parallelizing a Modular Program the SET Way
SET is a unique technology that significantly reduces the complexity of parallelizing modular programs. SET works "In Between" modules, inter-

cepting command flow in the modules' link. By providing an independent infrastructure layer (Pooch clustering solution) that covers partitioning and mapping data to processors, and providing internal MPI-based communication structure to processors that handle the parallel tasks, a great deal of complexity is taken away from parallelizing the sequential code. In addition, there are no modifications required for the original sequential code since SET can automatically provide its own instances of modules required to perform the parallel operations, without the sequential code even being "aware" of the parallelization. Porting a sequential program to work with SET is done by relatively simple glue code, where SET is instructed to intercept commands that flow between modules, depending on the command type.

Figure 1: Parallelizing Wolfram Research's Mathematica, where Semath modules are the silver spheres.


FRONT END
Supercomputing Engine Technology - SET
- Applied to *Mathematica*®
KERNEL
KERNEL
KERNEL
KERNEL
KERNEL
SEM - The Supercomputing Engine for Mathematica is an application of SET to parallelize Wolfram Research's *Mathematica*. Wolfram's code isn't even "aware" it is running in parallel, yet it provides significant speedup.
Patent Pending

SET Proof of Concept - Parallelizing Wolfram Research's Mathematica

As a SET proof of concept, we have decided to provide Wolfram Research's Mathematica with the ability to have a supercomputing-like parallelization infrastructure (MPI-based all-to-all communication between its computational kernels). Mathematica is a proprietary program, which does not allowed any modification of its source code, and it is an excellent example of modular programming. Mathematica has three parts. It has a front end (FE) module for user interface, a back end (BE) module which performs the calculations (Kernel), and a communication link between the FE and BE, called MathLink.

Although Mathematica has a version of limited grid-based parallel computing (A relatively small subset of true supercomputing), we have instead decided to apply SET to Mathematica itself and turn it into a full-featured supercomputer application. The name we gave this effort was SEM - Supercomputing Engine for Mathematica.

We began by implementing an MPI library within the Mathematica environment, an industry first. Applying the paradigm of distributed-memory MPI to Mathematica, SET launches multiple instances of the Mathematica kernel, each under the control of an instance of SET's Semath module. (See Figure 1 in page 3, and Figure 2 in page 5). These Semath modules construct and use a low-level MPI network communications layer. Expressions transmitted from any kernel are intercepted by Semath, forwarded between Semath's using the low-level MPI, then recreated in the target kernel elsewhere on the cluster. For the Mathematica environment, this process creates the illusion that Mathematica is calling MPI, but in fact SET is transmitting the expression as data using the low-level MPI.

SEM intercepts commands from the FE and forwards them via the Semath's on the cluster to all the kernels. The kernels then perform their work and coordinate using MPI like modern supercomputers, and the FE can display the results.

SEM's API is divided into three categories:

- Low-level: Point-to-point transmissions, synchronous and asynchronous.
- Collective: Communications involving any subset of processors.
- High-level communications: Implement commonly used tasks, be-

haviors, and communications patterns present across parallel computing.

"Everything is an Expression" in Mathematica, so subroutines, functions, graphics, sounds, and equations can be sent via MPI, not just data. An industry first, SEM provides Mathematica with unprecedented capabilities thanks to SET. The porting effort of Mathematica to SET took one (1) skilled man/month.

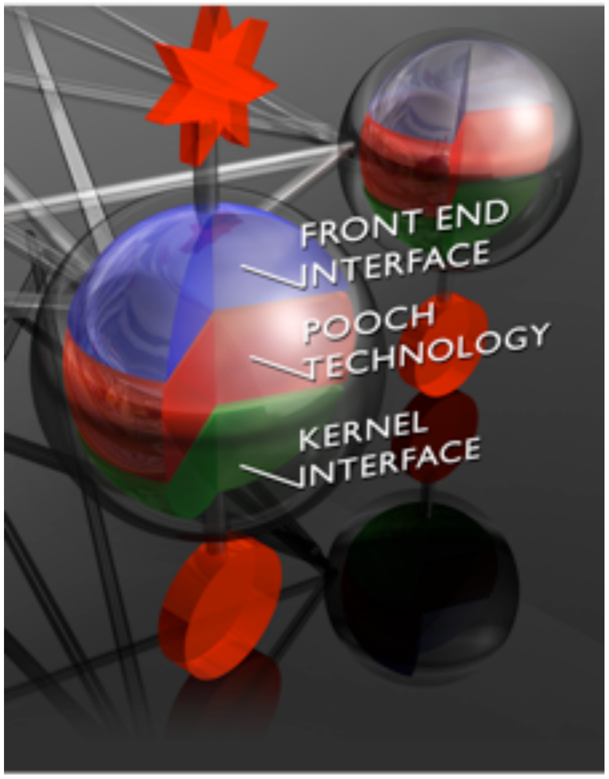


Figure 2: Semath modules Structure

Additional Implementations and Future of SET

Applying SET to Mathematica (SEM) is only one example of what SET can do. In embedded systems or large programs, for example, a single computational module in the work flow can slow the whole system due to slow processing speeds. This module can be locally ports to SET for parallelization, therefore improving performance of the whole program or workflow, without modification to the module's source code. Other implementations of SET can accelerate video compression, Finite Element Method calculations, etc.

With additional capital investments, we seek to take SET to the next level and provide software vendors and researchers with a toolkit which will provide a relatively quick and efficient way to port their sequential software to any parallel platform - multi-core machines or a supercomputer - without modification to the sequential code. This can simplify and speed up development of new programs tremendously since there is no need to design and debug a parallel version of a program from scratch. Making a sequential program work flawlessly is a very important first step. With the application of SET, the sequential program (or particularly slow portions of it) can be then easily ported to SET and take advantage of multi-core or supercomputing in a fraction of time it would take to parallelize it the standard way.