As we look toward the future, we expect greater capability along with disruptive changes in high performance computing systems

- Extreme levels of concurrency
  - Very high node and core counts
  - Increasingly deep memory hierarchies
- Additional complexities
  - Hybrid architectures
  - Manycore, GPUs, multithreading
  - Relatively poor memory latency and bandwidth
  - Challenges with fault resilience
  - Must conserve power – limit data movement
  - New (not yet stabilized) programming models
- Etc.

New capabilities will enable new computational science opportunities

- Enough computational power to enable
  - Multirate, multiscale, multicomponent, multiphysics simulations
  - Uncertainty quantification and sensitivities for all simulations
  - Simulations involving stochastic quantities
  - Optimization over full-featured simulations
  - Coupling of simulations and data analytics

Beyond interpretive simulations … working toward predictive science

Multiphysics applications share many common characteristics

- Coupling often done through combining existing codes that simulate subsystems
- Protects investment in V&V
- Protects intellectual property
- Allows exploitation of model-specific characteristics within methods
- Allows for parallelism over submodels

Common challenges
- Choice of coupling scheme to ensure stable and accurate solutions
- V & V and UQ of coupled system
- Engineering software for efficiency of computation and code maintenance

Multiphysics challenges ... the study of ‘and’

“We often think that when we have completed our study of one we know all about two, because ‘two’ is ‘one and one.’ We forget that we still have to make a study of ‘and.’”

- Sir Arthur Stanley Eddington (1892–1944), British astrophysicist
Operator splitting can destabilize multiphysics

---

Example from Estep et al. (2008)  \( u + \lambda u^2 = u^m \),  \( u(0) = u_0, t > 0 \)

\[
u(t) = \frac{u_0 \exp(-\lambda t)}{1 + \frac{v}{p} \left( \exp(-\lambda t) - 1 \right)}
\]

50 time steps, phase 1 subcycled inside phase 2

\( v_0(t) = \frac{u_0}{1 - \frac{v_0}{p} (t - t_0)} \) \( v_0(t) = v_0(t_{k+1}) \exp(-\lambda (t - t_0)) \) \( \Delta t = \frac{u_0}{1 - \frac{v_0}{p} \exp(-\lambda \Delta t)} \)

---

Multirate integrators will aid applications coupling more physics

---

Variable Partitioned:

\[
y' = f(y)
\]

where

\[
y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_3 \end{bmatrix}
\]

Fast

Slow

Additive:

\[
y' = f_{\text{fast}}(y) + f_{\text{slow}}(y)
\]

- Important for multiphysics
- Commonly time stepped using splitting + subcycling
- Less-well covered in literature

Received the most attention in literature...

---

Software aspects of additive case are simpler

- Partitioning of the problem is static
- Right-hand-side evaluation and Jacobian are the same
- Multirate does not add (much) additional complexity to MPI parallelism beyond multiphysics
  - None if do not need coupled implicit solve
- May require passing multiple snapshots of state between physics
  - Some methods require only one, e.g., RFSMR
- The question of “where to put the solve” in implicit methods is an open question

---

Things can go wrong with current methods: Operator splitting treatment can lose accuracy

---

Prototype for reaction-diffusion PDE:

\[
u \frac{\partial u}{\partial t} = \mu (u - 1) + D \frac{\partial^2 u}{\partial x^2} \]

BCs: \( u(l,t) = 1 \), \( u(l,t) = 0 \)

- A linear one-dimensional PDE with exact separation-of-variables solution
- Reaction-first (“RD”) splitting:
  - Diffusion-first (“D”) splitting:

\[
u = u^*, \quad \frac{u^* - u}{\Delta t} = 0, \quad \frac{u^* - u}{\Delta t} = \frac{u^* - u}{\Delta t} = 0
\]

- Result at \( \Delta t = 0 \), reaction rate \( k = 20 \), \( \Delta t = 0.01 \)

- “Base” is exact
- “Delayed” is 1st order backward Euler (eventually converges)
- Split schemes never converge to correct solution

---

Multirate integrators couple fast and slow components through interpolation/approximation

---

Can be via an explicit interpolation/extrapolation formula… (as in AMR applications)

- Using linear combinations of prior data, where the weights are derived directly from order conditions
  - e.g., Generalized additive Runge-Kutta methods and their multirate variants (Sandu and G"unther)

---

Application of explicit-explicit additive methods to MG2 microphysics shows the benefits of multirate

\[
\frac{\partial u}{\partial t} + \nabla \cdot [\mu u] = \left( \frac{1}{\alpha} \right) \text{core} + \left( \frac{1}{\beta} \right) \text{exp} + \left( \frac{1}{\gamma} \right) \text{acc}
\]

Morrison & Gettelson, 2009

\[
\frac{1}{\alpha} \text{order splitting with 10:1 subcycling requires ~36 hours to bring error below 2-norm of 1e-4} \\
\text{RFSMR 3rd-order (Schlegel et al.) requires ~40 minutes} \\
\text{RFSMR 4th-order developed by Jean Sexton will be added to SUNDIALS}
\]

Multirate gives benefits of subcycling, with the significant advantage of higher-order accuracy
Multirate integrators will coordinate use of multiple mathematical capabilities

- For implicit approaches, will need
  - Function and Jacobian
  - Nonlinear solvers
  - Linear solvers
- Will need these for subcomponents of the full system
- Stability and accuracy are dependent on adequately capturing the coupling between subsystems

Flexible multiphysics software is essential

Must fundamentally rethink approaches to multiphysics models, algorithms, and solvers with attention to data motion, data structure conversion, and overall application design.

**Challenges:**

- Enable introduction of new models, algorithms, and data structures
- Address CS issues for coupled codes
  - mapping codes to machine topologies
  - load balancing, resilience, etc.
- Competing goals of software interface stability and software reuse with the ability to innovate algorithmically and develop new physical models

Scientific software development encounters challenges from both the technical and sociological arenas

**Technical**

- All parts of the cycle can be under research
- Requirements change throughout the lifecycle as knowledge grows
- Importance of reproducibility
- Verification complicated by floating point representation
- Real world is messy, so is the software

**Sociological**

- Compelling priorities and incentives
- Limited resources
- Perception of overhead with deferred benefit
- Need for interdisciplinary interactions

Some prior work has investigated detection of coupling “tightness” and stability concerns with splitting

- Use of Jacobian block characterizations to detect strength of coupling
  - Analysis of coupling error for block Gauss-Seidel handling of two coupled modules using Jacobian information
- Use of adjoints and a priori error estimates to determine time steps at which operator split reaction-diffusion equations will be stable
- None of these methods has been developed into mature software

Increasing complexity of future computational science problems leads to increasing complexity of software

There is an ecosystem imperative for the math software, computational science, and high performance computing communities to improve dialogue

**Classic application development approach:**

- Application developers write most code; source code considered private
- Occasionally use libraries, typically only those “baked into” the operating system
- Portability challenges, unmanaged disruptions

**Ecosystem-based application development approach:**

- Application developers write glue code and unique functionality
- Source code includes substantial 3rd party packages
- Risks:
  - Dependent on portability of 3rd party code
  - Upgrades can be disruptive
- Opportunities:
  - 3rd party improvements are free
  - Portability is seamless

Science through computing is only as good as the software that produces it.
Software libraries are not enough: the xSDK effort was started to address challenges with using multiple libraries at once

Next-generation scientific simulations require combined use of independent packages

- Installing multiple independent software packages is error prone
  - Need consistency of compiler (+version, options), 3rd-party packages, etc.
  - Namespace and version conflicts make simultaneous build/link of packages difficult
- Multilayer interoperability requires careful design

Prior to xSDK effort, could not build required libraries into a single executable due to many incompatibilities

xSDK history: Work began in ASC/RBER partnership, IDEAS project (Sept 2014)

Needed for multiscale, multiphysics integrated surface-subsurface hydrology models

The extreme-scale software development kit (xSDK) is trying to address some of these issues for many US Department of Energy math libraries

xSDK community policies: Help address challenges in interoperability and sustainability of software developed by diverse groups at different institutions

**xSDK compatible package:** must satisfy the mandatory xSDK policies
- Also have recommended policies, which currently are encouraged but not required

**Impact:**
- Improved code quality, usability, access, sustainability
- Foundation for work on interoperability and portability

xSDK member package:
1. Must be an xSDK-compatible package, and its uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.
2. The package must support production use.
3. Be buildable using 64 bit pointers. 32 bit is optional.
4. Allow installing, building, and linking against an outside copy of external software.
5. Have no hardwired print or IO statements.
6. Have an accessible repository (not necessarily publicly available).
7. Respect system resources and settings made by other previously called packages.
8. Provide a comprehensive test suite.
10. Provide a runtime API to return the current version number of the software.
11. Provide an accessible repository (not necessarily publicly available).
12. Come with an open source license.
13. Provide a limited and well defined symbol, macro, library, and include file name space.
14. Have no hardwired print or IO statements.
15. Provide an accessible repository (not necessarily publicly available).
16. Allow installing, building, and linking against an outside copy of external software.
17. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.
18. Be buildable using 64 bit pointers. 32 bit is optional.
19. All xSDK compatibility checks should be sustainable.
20. The package must provide production-quality installation compatible with the xSDK install tool and xSDK metapackage.

The xSDK community comes from developers of numerous numerical libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salah Balay</td>
<td>[ANL] Release lead</td>
<td>MagDA, MFEM, PETSc, SUNDIALS, SuperLU, Trilinos</td>
</tr>
<tr>
<td>Ross Bartlett</td>
<td>[SNL] Member packages</td>
<td>hypre, MAGMA, PETSc, SUNDIALS, SuperLU, Trilinos</td>
</tr>
<tr>
<td>Mark Barrièr</td>
<td>[ORNL] Leads for xSDK testing</td>
<td>Salah Balay, ALCF, Mark Barrièr, ORNL, SNL, UCLA</td>
</tr>
<tr>
<td>Aaron Fischer</td>
<td>[NERSC] Space lead</td>
<td>Salah Balay, ALCF, Mark Barrièr, ORNL, SNL, UCLA</td>
</tr>
</tbody>
</table>

xSDK release lead:
- Salah Balay: ALCF
- Mark Barrièr: ORNL
- Aaron Fischer: NERSC
- Keita Teranishi: overall

xSDK community policies: These are useful practices for any scientific simulation code

- ML. Support xSDK community GNU Autotools or CMake options.
- ML. Provide a comprehensive test suite.
- ML. Employ user-provided MPI communicator.
- M4. Give best effort at portability to key architectures.
- M5. Provide a documented, reliable way to contact the development team.
- M5. Respect system resources and settings made by other previously called packages.
- MT. Come with an open source license.
- M5. Provide a runtime API to return the current version number of the software.
- M6. Use a limited and well-defined symbol, macro, library, and include file name space.
- M7. Provide an accessible repository (not necessarily publicly available).
- M8. Allow installing, building, and linking against an outside copy of external software.
- M9. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.
- M10. Be buildable using 64-bit pointers. 32-bit is optional.
- M11. All xSDK compatibility checks should be sustainable.
- M12. The package must provide production-quality installation compatible with the xSDK install tool and xSDK metapackage.

xSDK recommended community policies

Also recommended policies, which currently are encouraged but not required:

- R1. Have a public repository.
- R2. Possible to run test suite under valgrind in order to test for memory corruption issues.
- R3. Adopt and document consistent system for error conditions/exceptions.
- R4. Free all system resources it has acquired as soon as they are no longer needed.
- R5. Provide a mechanism to export ordered list of library dependencies.

Through the Exascale Computing Project, the xSDK is also facilitating greater interoperability between member packages

- **PETSc:**
  - hypre, SuperLU, Trilinos linear solvers
  - SUNDIALS time integrators
- **Trilinos:** hypre, SuperLU, PETSc linear solvers
- **HYPRE:**
  - SuperLU for coarse grid solvers
  - Planned: interoperability with PETSc and Trilinos matrix structures
- **SUNDIALS:** SuperLU, hypre, PETSc, MAGMA, and more
- **MFEM:** PETSc solvers, SUNDIALS time integrators

The xSDK community is trying to address some of these issues for many US Department of Energy math libraries

xSDK: [https://xsdk.info](https://xsdk.info)

The xSDK community comes from developers of numerous numerical libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salah Balay</td>
<td>[ANL] Release lead</td>
<td>MagDA, MFEM, PETSc, SUNDIALS, SuperLU, Trilinos</td>
</tr>
<tr>
<td>Ross Bartlett</td>
<td>[SNL] Member packages</td>
<td>hypre, MAGMA, PETSc, SUNDIALS, SuperLU, Trilinos</td>
</tr>
<tr>
<td>Mark Barrièr</td>
<td>[ORNL] Leads for xSDK testing</td>
<td>Salah Balay, ALCF, Mark Barrièr, ORNL, SNL, UCLA</td>
</tr>
<tr>
<td>Aaron Fischer</td>
<td>[NERSC] Space lead</td>
<td>Salah Balay, ALCF, Mark Barrièr, ORNL, SNL, UCLA</td>
</tr>
</tbody>
</table>

xSDK release lead:
- Salah Balay: ALCF
- Mark Barrièr: ORNL
- Aaron Fischer: NERSC
- Keita Teranishi: overall

xSDK community policies: These are useful practices for any scientific simulation code

- ML. Support xSDK community GNU Autotools or CMake options.
- ML. Provide a comprehensive test suite.
- ML. Employ user-provided MPI communicator.
- M4. Give best effort at portability to key architectures.
- M5. Provide a documented, reliable way to contact the development team.
- M5. Respect system resources and settings made by other previously called packages.
- MT. Come with an open source license.
- M5. Provide a runtime API to return the current version number of the software.
- M6. Use a limited and well-defined symbol, macro, library, and include file name space.
- M7. Provide an accessible repository (not necessarily publicly available).
- M8. Allow installing, building, and linking against an outside copy of external software.
- M9. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.
- M10. Be buildable using 64-bit pointers. 32-bit is optional.
- M11. All xSDK compatibility checks should be sustainable.
- M12. The package must provide production-quality installation compatible with the xSDK install tool and xSDK metapackage.

xSDK recommended community policies

Also recommended policies, which currently are encouraged but not required:

- R1. Have a public repository.
- R2. Possible to run test suite under valgrind in order to test for memory corruption issues.
- R3. Adopt and document consistent system for error conditions/exceptions.
- R4. Free all system resources it has acquired as soon as they are no longer needed.
- R5. Provide a mechanism to export ordered list of library dependencies.

Through the Exascale Computing Project, the xSDK is also facilitating greater interoperability between member packages

- **PETSc:**
  - hypre, SuperLU, Trilinos linear solvers
  - SUNDIALS time integrators
- **Trilinos:** hypre, SuperLU, PETSc linear solvers
- **HYPRE:**
  - SuperLU for coarse grid solvers
  - Planned: interoperability with PETSc and Trilinos matrix structures
- **SUNDIALS:** SuperLU, hypre, PETSc, MAGMA, and more
- **MFEM:** PETSc solvers, SUNDIALS time integrators

The xSDK community comes from developers of numerous numerical libraries

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Salah Balay</td>
<td>[ANL] Release lead</td>
<td>MagDA, MFEM, PETSc, SUNDIALS, SuperLU, Trilinos</td>
</tr>
<tr>
<td>Ross Bartlett</td>
<td>[SNL] Member packages</td>
<td>hypre, MAGMA, PETSc, SUNDIALS, SuperLU, Trilinos</td>
</tr>
<tr>
<td>Mark Barrièr</td>
<td>[ORNL] Leads for xSDK testing</td>
<td>Salah Balay, ALCF, Mark Barrièr, ORNL, SNL, UCLA</td>
</tr>
<tr>
<td>Aaron Fischer</td>
<td>[NERSC] Space lead</td>
<td>Salah Balay, ALCF, Mark Barrièr, ORNL, SNL, UCLA</td>
</tr>
</tbody>
</table>

xSDK release lead:
- Salah Balay: ALCF
- Mark Barrièr: ORNL
- Aaron Fischer: NERSC
- Keita Teranishi: overall

xSDK community policies: These are useful practices for any scientific simulation code

- ML. Support xSDK community GNU Autotools or CMake options.
- ML. Provide a comprehensive test suite.
- ML. Employ user-provided MPI communicator.
- M4. Give best effort at portability to key architectures.
- M5. Provide a documented, reliable way to contact the development team.
- M5. Respect system resources and settings made by other previously called packages.
- MT. Come with an open source license.
- M5. Provide a runtime API to return the current version number of the software.
- M6. Use a limited and well-defined symbol, macro, library, and include file name space.
- M7. Provide an accessible repository (not necessarily publicly available).
- M8. Allow installing, building, and linking against an outside copy of external software.
- M9. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.
- M10. Be buildable using 64-bit pointers. 32-bit is optional.
- M11. All xSDK compatibility checks should be sustainable.
- M12. The package must provide production-quality installation compatible with the xSDK install tool and xSDK metapackage.

xSDK recommended community policies

Also recommended policies, which currently are encouraged but not required:

- R1. Have a public repository.
- R2. Possible to run test suite under valgrind in order to test for memory corruption issues.
- R3. Adopt and document consistent system for error conditions/exceptions.
- R4. Free all system resources it has acquired as soon as they are no longer needed.
- R5. Provide a mechanism to export ordered list of library dependencies.
Software productivity and sustainability plans lay out processes that contributors are expected to follow

- Requirements for design documents and project review of new features
- Establishes a workflow for submission of new code and its review
- Establishes documentation requirements
- Describes testing practices
- Establishes practices for external package usage and testing
- Prevents conflicting features, poorly designed, and untested code
- Living document, expected to be updated and changed as software evolves

High performance currently requires adopting one of numerous programming models reducing code portability

- MPI
- OpenMP
- PThreads
- OpenMP 4.5
- OpenACC
- CUDA
- Hybrids of any and all of the above models

- Numerous packages are being developed to provide a single interface to all of these with no clear winner
  -- Kokkos (SNL)
  -- RAJA (LLNL)
  -- OCCA (VA Tech)

The xSDK is starting to address how to develop portability for node-level resource management

Need to develop ways to pass data and runtime resource information
- Between applications and libraries and among libraries
- Should not impose a particular threading model
- Should obtain memory layout via a basic C-like API
- Need interface support to allocate memory

Some options:
- Encapsulate data use and sharing as much as possible
- Employ portability layers, such as Kokkos or RAJA (like early days of distributed parallelism when we had the NX, CMMD, etc. proprietary libraries giving way to Paragraph, PVM, and ultimately MPI)
- Use new features of MPI, such as split communicator and MPI Window, to create subgroups of ranks such that each are able to share memory

A report assessing what libraries currently do and ways to go forward has been developed and will soon be available (effort lead by U. Yang (LLNL)).

Summary: Coupled Systems, Numerical Libraries, and High Performance Computing come together in a scientific software ecosystem

- Exascale opportunities give rise to unprecedented computational opportunities
- These opportunities present numerous challenges:
  -- Numerics need to be both accurate and stable across many scales
  -- Software costs are rising along with software complexity
  -- On-node programming models are changing rapidly
- Multirate numerical methods may help in addressing loss of accuracy and stability in the time integration
- Software libraries can provide highly specialized numerical methods easing the algorithm burden for application developers
- xSDK software policies provide a strong foundation for integration of multiple libraries and applications
- Portability layers help ease transition between programming models

Acknowledgements

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the U.S. Department of Energy Office of Advanced Scientific Computing Research and National Nuclear Security Administration.

computation.llnl.gov/casc/sundials

Support for this work was provided through the Exascale Computing Project (ECP), a collaborative effort of two DOE organizations – the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable esascle ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, to support the nation’s esascle computing imperative.