Build Times on Modern Heterogeneous Systems:

Steps towards a faster future

James Elliott and Johannes Doerfert

jjellio@sandia.gov doerfert1@llnl.gov

Trilinos User Group

October 23, 2024

Albuquerque, NM

Summary

- Building on heterogenous systems … why is it so expensive?
- Shared vs Static a gentle overview
- Case study, building Trilinos for Nvidia H100s and AMD MI300As
- Relocatable Device Code (RDC)
- Challenges with RDC on modern systems
- Looking forward – active engagements and future collaborations
- Conclusion/Discussion

Heterogenous Builds

- Part of Cost from portability
  - #define KOKKOS_INLINE_FUNCTION __device__ __host__ inline
- To support host/device execution many (most) functions get marked as __device__ __host__
  - Implies the compiler must generate both host and device object code
  - Can require multiple reads/writes of the same file, e.g., nvcc invokes g++ multiple times per file
- Other differences, but improving file IO can result in 2x compile time speedups (more later)
- Compiler Differences …
  - Cuda (nvcc) is GNU based, ROCM (amdclang) is LLVM based. LLVM also supports targeting Cuda.

Linkage: Shared and Static

- Linkage determines how the resulting executable or libraries are used
- **Shared Linkage**
  - Object code are compiled into shared (dynamic) libraries
  - At runtime, the dynamic loader (aka, dlopen / dlload) are responsible for loading symbols on-demand from shared libraries linked to your binary.
  - Can be punishing on filesystems (if you have many shared libraries)…
    - On large MPI launches using something like NFS (Network File System) for a filesystem … 'look like' *denial of service attacks.*
    - Hundreds (or thousands of processes) concurrently ask their node to load a file
    - On the Sierra supercomputer this lead to tools made to copy all shared libs for binary, into a parallel filesystem or locally on the node (e.g., Spindle from LLNL)

Linkage: Shared and Static

- Linkage determines how the resulting executable or libraries are used
- **Static Linkage**
  - Object code are compiled into static archives
  - At link time, the linker must resolve all symbols the binary requires
  - That is, find symbols in the archive, unpack the archive, and link the actual object files into the executable. (archives are mostly a container for object files with an index)
  - The binary will contain the actual executable code that was contained in the archive (binaries get large!)
  - No symbol resolution is needed for statically linked binaries, as all symbols are found at link time.

Linkage: Shared and Static

- Linkage determines how the resulting executable or libraries are used
- **Relocatable Device Code** (requires static)
  - In both Nvidia and AMD semantics, GPU functions must only reference data/symbols that are within the same compilation unit (e.g., a file)
  - **Virtual functions** create a problem – if the base class is defined else where, this would prohibit using OO design ideas for GPU kernels
  - RDC if enabled, allows a GPU function to resolve symbols outside the current compilation unit
- Host code is relocatable by default
- Vendors implement RDC differently
  - AMD implements RDC as an LLVM “link time optimization”

Case Study: Cuda on H100 and ROCM on MI300A

Setup

- Build Trilinos with ENABLE_ALL_PACKAGES=ON (turning off deprecated)
  - Disable unit test/examples
  - Enable Proxies: Panzer's **MiniEM** (EMPIRE) and Ifpack2's **blockTriDiagContainer** (SPARC)
- Enable the standard TPL (third party libraries) Sandia codes depend on (HDF5, NetCDF, Parallel NetCDF, Boost, METIS, ParMETIS, CGNS, LAPACK, BLAS, MPI)
- Enable Cuda or HIP
- Enable vendor TPLs for BLAS, Solvers (dense) and Sparse (cublas, cusolver, cusparse)
- Perform builds for shared, static and relocatable device code
- Creates about 3500 targets

Case Study: Cuda on H100 and ROCM on MI300A

Build Hardware

- Build on Saphire Rapids (112 cores) or quad Mi300A (96 cores)
- Build on local SSD (for Saphire Rapids) and on Ramdisk for Mi300A
  - Should minimize the impact of disk usage
- Use ninja and invoke with no parallelism constraint (e.g., just run ninja)

Versions

- Cuda 11.8 with GCC 11.2.1, AMD ROCM 6.1.2 (llvm 17 based toolchain)

Build Statistics

- Track total time (qualitative measure)
- Track each file's compile/link time
  - E.g., the time to link an app is measured directly
  - Uses tooling in Trilinos (Build Stats), wraps each compile/link step with a timer

Quality of Life … how much longer will this take!

Total Build Time

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		24m 2s	13m 1s
Static	9m 34s	25m 23s	13m 33s
RDC			**44m 22s**

Time to Link MiniEM

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		1.5s	0.95s
Static	43.99s	82.52s	7.09s
RDC			**32m 8s**

Not All Linkers are Created Equal … GNU ld vs LLVM lld

Total Build Time – CPU Utilization Observed in Parenthesis

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		24m 2s	13m 1s
Static	9m 34s	25m 23s	13m 33s
RDC			**44m 22s**

Time to Link MiniEM

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		1.5s (97%)	0.95s (113%)
Static	43.99s (99%)	82.52s (99%)	7.09s (604%)
RDC			**32m 8s (101%)**

AMD+RDC time is slow, partly because RDC is serializing the build
It's a link time optimization (different approach than Nvidia)

Linking My app takes longer than the standard work day.

Time to Link MiniEM

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		1.5s (97%)	0.95s (113%)
Static	43.99s (99%)	82.52s (99%)	7.09s (604%)
RDC			**32m 8s (101%)**

- What can be done?
  - Collab with AMD, Collab with LLNL (who has an LLVM developer!)
  - RDC implemented at LTO (link time optimization) is clever, but as-is is costly
  - **There is hope** – we see concurrent link of GPU code via LLVM lld (**604%**) with the static build
    - This hints that if AMD can improve how RDC fits into LLVM's linking ecosystem, we could potentially have performant links that also add code optimization
- Exploring "Thin" Link Time Optimization
  - AMD RDC is "Full" LTO

First Fruits of Collaborating on Link Times

Total Build Time – CPU Utilization Observed in Parenthesis

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		24m 2s	13m 1s
Static	9m 34s	25m 23s	13m 33s
RDC		27m 36s	**44m 22s**
RDC+ThinLTO		-	**18m 43s**

Time to Link MiniEM

	Host Only	Cuda on Sapphire Rapids	HIP on Mi300A
Shared		1.5s (97%)	0.95s (113%)
Static	43.99s (99%)	82.52s (99%)	7.09s (604%)
RDC		2m 40s (100%)	**32m 8s (101%)**
RDC+ThinTLO		-	**6m 32s (269%)**

Nearly 5x improvement in RDC link times with AMD by getting Thin LTO to work

Case Study: Cuda on H100 and ROCM on MI300A

Nvidia link 160.22s vs AMD 392.59s …. That's more than 2x faster!

… yes and no

- LTO builds are different from regular builds… with LTO, files are compiled into intermediate code (the .o file does not executable code)
- At link, all the intermediate code processed into one final object code
- E.g., Total build time

	Total Build Time	Executable Link Time
Cuda RDC	27m 36s	2m 40s
AMD RDC	44m 22s	32m 8s
AMD RDC+ThinTLO	18m 43s	6m 32s

- Qualitatively, the future isn't so murky
  - Still, RDC presents a problem testing … link time is per-executable

Summary

- I have evaluated build times and link times with
  - GCC 11.2.1 with Cuda 11.8 and ROCM 6.1.2
  - I built a representative Trilinos and linked the proxy app MiniEM
- Initially, Cuda builds were nearly 2x faster (Total time) for Relocatable Device Code
- Exploring advanced/modern linking technology in the LLVM toolchain …

	Total Build Time	Executable Link Time
Cuda RDC	**27m 36s**	2m 40s
AMD RDC	44m 22s	32m 8s
AMD RDC+ThinTLO	**18m 43s**	6m 32s

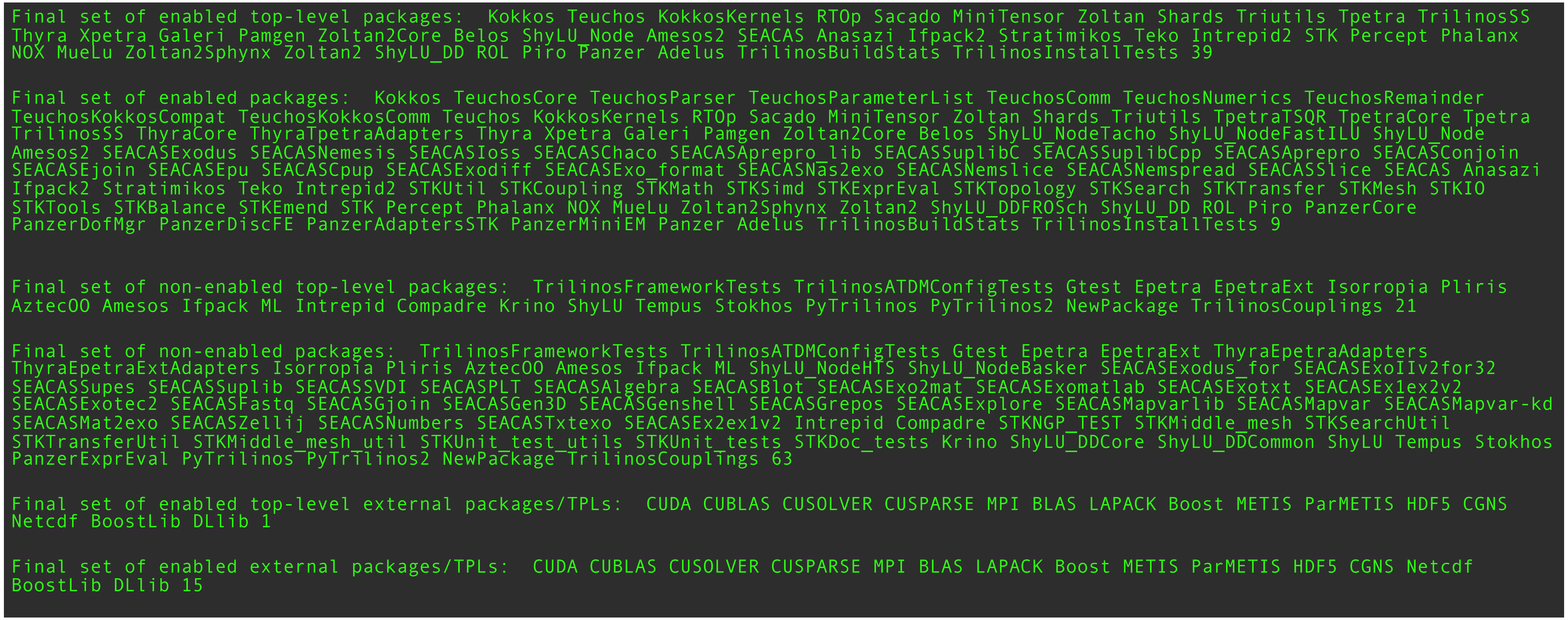
- We achieve a faster overall build time, while also seeing the link time of a complex binary improve by nearly **5x**
- Future (ongoing) work with AMD hopes to further parallelize the link step (NDA'd)

Questions : James Elliott <jjellio@sandia.gov>

Acknowledgement

“Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan.”

Trilinos Packages and TPLs (example)



```
Final set of enabled top-level packages:  Kokkos Teuchos KokkosKernels RTOp Sacado MiniTensor Zoltan Shards Triutils Tpetra TrilinosSS
Thyra Xpetra Galeri Pamgen Zoltan2Core Belos ShyLU_Node Amesos2 SEACAS Anasazi Ifpack2 Stratimikos Teko Intrepid2 STK Percept Phalanx
NOX MueLu Zoltan2Sphynx Zoltan2 ShyLU_DD ROL Piro Panzer Adelus TrilinosBuildStats TrilinosInstallTests 39

Final set of enabled packages:  Kokkos TeuchosCore TeuchosParser TeuchosParameterList TeuchosComm TeuchosNumerics TeuchosRemainder
TeuchosKokkosCompat TeuchosKokkosComm Teuchos KokkosKernels RTOp Sacado MiniTensor Zoltan Shards Triutils TpetraTSQR TpetraCore Tpetra
TrilinosSS ThyraCore ThyraTpetraAdapters Thyra Xpetra Galeri Pamgen Zoltan2Core Belos ShyLU_NodeTacho ShyLU_NodeFastILU ShyLU_Node
Amesos2 SEACASExodus SEACASNemesis SEACASIoss SEACASChaco SEACASAprepro_lib SEACASSuplibC SEACASSuplibCpp SEACASAprepro SEACASConjoin
SEACASEjoin SEACASEpu SEACASCpup SEACASExodiff SEACASExo_format SEACASNas2exo SEACASNemslice SEACASNemspread SEACASSlice SEACAS Anasazi
Ifpack2 Stratimikos Teko Intrepid2 STKUtil STKCoupling STKMath STKSimd STKExprEval STKTopology STKSearch STKTransfer STKMesh STKIO
STKTools STKBalance STKEmend STK Percept Phalanx NOX MueLu Zoltan2Sphynx Zoltan2 ShyLU_DDFROSch ShyLU_DD ROL Piro PanzerCore
PanzerDofMgr PanzerDiscFE PanzerAdaptersSTK PanzerMiniEM Panzer Adelus TrilinosBuildStats TrilinosInstallTests 9


Final set of non-enabled top-level packages:  TrilinosFrameworkTests TrilinosATDMConfigTests Gtest Epetra EpetraExt Isorropia Pliris
AztecOO Amesos Ifpack ML Intrepid Compadre Krino ShyLU Tempus Stokhos PyTrilinos PyTrilinos2 NewPackage TrilinosCouplings 21

Final set of non-enabled packages:  TrilinosFrameworkTests TrilinosATDMConfigTests Gtest Epetra EpetraExt ThyraEpetraAdapters
ThyraEpetraExtAdapters Isorropia Pliris AztecOO Amesos Ifpack ML ShyLU_NodeHTS ShyLU_NodeBasker SEACASExodus_for SEACASExoIIv2for32
SEACASSupes SEACASSuplib SEACASSVDI SEACASPLT SEACASAlgebra SEACASBlot SEACASExo2mat SEACASExomatlab SEACASExotxt SEACASEx1ex2v2
SEACASExotec2 SEACASFastq SEACASGjoin SEACASGen3D SEACASGenshell SEACASGrepos SEACASExplore SEACASMapvarlib SEACASMapvar SEACASMapvar-kd
SEACASMat2exo SEACASZellij SEACASNumbers SEACASTxtexo SEACASEx2ex1v2 Intrepid Compadre STKNGP_TEST STKMiddle_mesh STKSearchUtil
STKTransferUtil STKMiddle_mesh_util STKUnit_test_utils STKUnit_tests STKDoc_tests Krino ShyLU_DDCore ShyLU_DDCommon ShyLU Tempus Stokhos
PanzerExprEval PyTrilinos PyTrilinos2 NewPackage TrilinosCouplings 63

Final set of enabled top-level external packages/TPLs:  CUDA CUBLAS CUSOLVER CUSPARSE MPI BLAS LAPACK Boost METIS ParMETIS HDF5 CGNS
Netcdf BoostLib DLlib 1

Final set of enabled external packages/TPLs:  CUDA CUBLAS CUSOLVER CUSPARSE MPI BLAS LAPACK Boost METIS ParMETIS HDF5 CGNS Netcdf
BoostLib DLlib 15
```