RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030 PROGRAMME DE RECHERCHE *NUMÉRIQUE POUR L'EXASCALE*

Guix-Based Deployment and Unified Multi-GPU Support for NumPEx Proxy-Apps

A. Dauteuil, F. Kpadonou

February 25, 2026

General Assembly Exa-DI

Plan

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Guix developments

Our Guix-channel: `guix-numpex`

The CDT uses Guix as its primary tool for deploying and managing the PC5 proxy-application stack.

`https://gitlab.inria.fr/numpex-pc5/guix-numpex`

`guix-numpex` channel

- Includes most of the applications involved in GT 1-3
- Reproducible, traceable builds across HPC platforms

Apologize to Enabling Team : we will contribute to Guix-science ...

Packaged Applications Across Working Groups

Working Group	Packaged Applications
GT1 - High Order Schemes	proxy-GEOS, proxy-FUn (CUDA & HIP)
GT2 - Unstructured Mesh	Arcane, Sharc
GT3 - Block-structured AMR	Dyablo (AthenaK and/or Castro coming soon)

Focus: proxy-GEOS and multiplication of configurations

What is proxy-GEOS?

Proxy application for wave propagation simulations (2D/3D acoustic wave equation)

- Multiple programming models: OpenMP, RAJA, **Kokkos**
- Supported on AMD & Nvidia GPUs

The packaging challenge

Goal: Provide ready-to-use packages covering all configurations

- GPU backends (CUDA, HIP) $\times$ architectures (V100, A100, MI250...)
- Users should not need package transformations or manual tweaks
- Combinatorial explosion of package variants

$\Longrightarrow$ Exploration of multiple GPU applications

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Multiple GPUs application

Nvidia GPUs Binary Compatibility

No binary compatibility across GPU generations

A binary compiled for one GPU architecture will not run on another generation

- Different instruction sets and capabilities
- Example: V100 (Volta) $\not\to$ A100 (Ampere) $\not\to$ H100 (Hopper)

Within-generation compatibility

Limited forward-compatibility within the same generation

- Example: A100 binary $\to$ A6000 (both Ampere)

Question: How to build once and support multiple generations?

Step 1: Virtual Architecture

What is a virtual architecture?

- **Not an actual GPU architecture**
- Defines a set of generic instructions and capabilities
- Used to generate **PTX assembly** (Parallel Thread Execution)

Role in compilation

PTX serves as intermediate representation to build application binaries for specific GPU architectures

Syntax

```
--gpu-architecture=compute_XX
```

- Example: `compute_80` for Ampere generation

Step 2: JIT or multiple CUDA Binaries Mechanism

The GPU compatibility mechanism depends on **when** the real architecture is specified.

multiple CUDA Binaries (cubins) (compile-time)

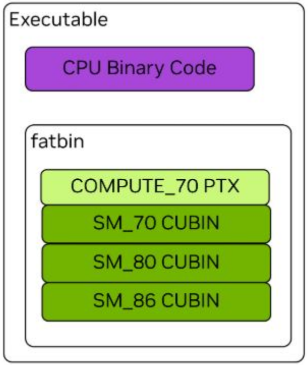
- Generate multiple binary versions at compile time
- One specific binary per targeted GPU model
- Real GPU architectures must be known at compile time

Just-In-Time (JIT) Compilation (runtime)

- GPU code generated at runtime when driver detects the architecture
- Induces startup overhead (mitigated by caching)
- Forward-compatible with future GPU models

Note: Both mechanisms can be combined (hybrid approach)

Fat Binary Structure



Source: NVIDIA Developer Blog
`https://developer.nvidia.com/blog/understanding-ptx-the-assembly-language-of-cuda-gpu-computing/`

What's inside?

- CPU binary code (host)
- Multiple CUBINs for targeted architectures
- PTX for JIT compilation
- Driver selects appropriate code at runtime

Compilation Setup for Binary compatibility support

Two key compilation options

`--gpu-architecture`: virtual architecture (Step 1)
`--gpu-code`: real architecture(s) or virtual for JIT (Step 2)

Example 1: Forward-compatible within generation

```
nvcc --gpu-architecture=compute_80 --gpu-code=sm_80 x.cu -o x.exe
```

Works on: A100 (sm_80), A6000 (Ampere 8.0+) — Fails on: V100 (older gen), H100 (no JIT)

Example 2: Hybrid (CUDA binary + JIT)

```
nvcc --gpu-architecture=compute_80 --gpu-code=sm_80,compute_80 x.cu
```

Works on: A100 (sm_80), H100 (sm_90) with JIT — Fails on: V100 (older gen)

Toy Use-Case: Matrix-Vector Multiplication

Testing methodology

Simple CUDA kernel to benchmark compilation mechanisms

- Matrix-vector product: 1000×1000 matrix
- Execution time averaged over 100 runs
- Tested on Ada (sm_89) GPU with CUDA 12.4

CUDA kernel

```
__global__ void mat_vect_prod(
    float* mat, float* vec,
    float* res, int n, int m) {
  int idx = blockIdx.x * blockDim.x
            + threadIdx.x;
  if (idx < n) {
    for (int i = 0; i < m; i++)
      res[idx] += mat[m*idx+i]
                  * vec[i];
  }
}
```

Comparison metrics

- Executable size
- Runtime performance
- Impact of multiple cubins vs JIT

multiple CUDA Binary: Results

Configuration	--gpu-arch	--gpu-code	Size	Time (s)
Fat80	compute_80	sm_80	812K	0.125
Fat89	compute_89	sm_89	812K	0.122
Fat80/80-86-89	compute_80	sm_80,sm_86,sm_89	824K	0.132
Fat80-86-89	compute_80	sm_80	824K	0.128
	compute_86	sm_86		
	compute_89	sm_89		

- Multiple architectures increase binary size slightly (+12K)
- Best performance obtained for the specific build of Ada89 (Fat89)
- Performance remains consistent across configurations

JIT Compilation: Results

Configuration	--gpu-arch	--gpu-code	Size	Time (s)	Time (s)[1]
JIT80	compute_80	compute_80	808K	0.134	0.14
JIT89	compute_89	compute_89	808K	0.137	0.15
JIT80 + Fat80/89	compute_80	compute_80, sm_89	812K	0.123	0.14
JIT89 + Fat89	compute_89	compute_89, sm_89	812K	0.12	0.136
JIT70 + Fat70/89	compute_70	compute_70, sm_89	812K	0.14	0.144
JIT70+Fat70/89 +	compute_70	compute_70, sm_89	824K	0.123	0.1375
JIT80+Fat80/89	compute_80	compute_80, sm_89			

[1] Cache disabled + JIT forced

Key observations

- Executable sizes remain constant among several configurations
- Execution times remain constant among several configurations
- JIT caching is effective (uncached: +5-10% overhead)

Application to proxy-GEOS

From toy example to real application

The multi-GPU mechanisms apply to the full proxy-GEOS stack

- Configure `CMAKE_CUDA_ARCHITECTURES` with explicit list: `70;80;86;89` $\Longrightarrow$ Hybrid approach using multiple cubins & JIT.
- Use GPU compiler (nvcc/hipcc) as main compiler
- Enable corresponding CMake language (CUDA or HIP)
- **Third-Party Libraries** (Kokkos, Raja,etc.) must also support multi-GPU compilation
  - Available features in Raja
  - Patched manually in Kokkos (No details given here)

proxy-GEOS: Test Configurations

Goal: Benchmark multi-GPU support on V100 (7.0) and A100 (8.0)

Test configurations

- `Hybrid70`: V100-based (`CMAKE_CUDA_ARCHITECTURES=70`) with JIT fallback
- `Hybrid80`: A100-based (`CMAKE_CUDA_ARCHITECTURES=80`) with JIT fallback
- `all-major_80`: All major architectures up to 8.0 (`CMAKE_CUDA_ARCHITECTURES="50;60;70;80"`)
- `all_86`: All architectures up to 8.6 (`CMAKE_CUDA_ARCHITECTURES="50;52;53;60;61;62;70;72;75;80;86"`)

proxy-GEOS: Benchmark Results

Config	Size	V100 (no cache)	A100 (no cache)	V100 (cached)	A100 (cached)
Hybrid80	3.2M	N/A	4.02s	N/A	2.51s
Hybrid70	3.2M	5.24s	4.08s	3.75s	2.54s
all-major_80	5.2M	5.21s	4.04s	3.75s	2.57s
all_86	9.9M	5.23s	4.04s	3.73s	2.60s

- Binary size increases with more architectures (3.2M $\rightarrow$ 9.9M)
- JIT caching reduces runtime by 30%
- Performance remains consistent across multi-GPU configurations

Impact on Guix Packaging

Before: Multiple packages

- `proxy-geos-kokkos-cuda-v100`
- `proxy-geos-kokkos-cuda-a100`
- `proxy-geos-kokkos-cuda-a40`
- `proxy-geos-kokkos-cuda-ada`
- `proxy-geos-kokkos-cuda-h100`

After: Single unified package

- `proxy-geos-kokkos-cuda-all`

```
Fatbin elf code:
arch = sm_75
arch = sm_80
arch = sm_86
arch = sm_89
arch = sm_90
```

Solution: One package, multiple GPUs supported

Result: Easy deployment for users, reduced maintenance for developpers

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

FRANCE 2030

PROGRAMME DE RECHERCHE
NUMÉRIQUE POUR L'EXASCALE

Conclusion

Conclusion

Multi-GPU build: Lessons learned

- Hybrid multiple cubins + JIT approach balances performance and flexibility
- One package replaces dozens of architecture-specific variants
- Extend the work for AMD GPUs

guix-numpex channel: Next steps

- Contribute back to **Guix-Science** channel

Thank you for your attention. Any Questions?

https://numpex-pc5.gitlabpages.inria.fr/wp2-co-design/doc-hub/multigpus-build/index.html

Appendix: Kokkos Patches for Multi-GPU Support

Why patch Kokkos?

Native Kokkos (v4.6.1) did not support multiple GPU architectures when using `KOKKOS_ENABLE_COMPILE_AS_CMAKE_LANGUAGE`

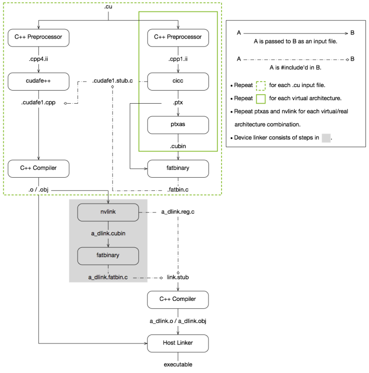
Key modifications in `kokkos_arch.cmake`:

- **Allow multiple architectures:** Remove fatal error when multiple `KOKKOS_ARCH_XX` are specified
- **Build architecture list:** Append architectures to `KOKKOS_CUDA_ARCHITECTURES` instead of overwriting
- **Propagate to CMake:** Set `CMAKE_CUDA_ARCHITECTURES` with semicolon-separated list

Additional fix in `Kokkos_DesulAtomicsConfig.hpp`:

- Define `KOKKOS_ARCH_XXX` macros based on runtime `__CUDA_ARCH__`
- Enables architecture-specific code paths within kernels

Appendix: Cuda compilation



CUDA Compilation Trajectory

Source: NVIDIA Documentation
`https://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/`